算法设计与分析

Computer Algorithm Design & Analysis

2024.12

王多强

QQ：1097412466

Chapter 24

Single-Source Shortest Paths

单源最短路径

本章主要内容：

1、Bellman-ford算法

2、Dijkstra算法

3、差分约束系统

1、最短路径问题

给定一个带权重的有向图 $G = (V, E)$ 和权重函数 $w: E \rightarrow R$。图中一条路径 $p = \langle v_0, v_1, \ldots, v_k \rangle$ 的权重 $w(p)$ 是构成该路径的所有边的权重之和：

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

从结点 u 到结点 v 的**最短路径权重**记为 $\delta(u, v)$，定义如下：

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\leadsto} v\} & \text{if there is a path from } u \text{ to } v \,, \\ \infty & \text{otherwise} \,. \end{cases}$$

- 具有**最小权重的路径**称为**最短路径。**
- **最短路径问题：**就是**求结点间的最短路径**，包括**路径权重**和**路径上的结点序列**。

◆ **单源点最短路径问题**：

给定一个图 $G = (V, E)$，找出从给定的**源点** $s \in V$ 到其它每个结点 $v \in V$ 的最短路径。

2、最短路径的最优子结构

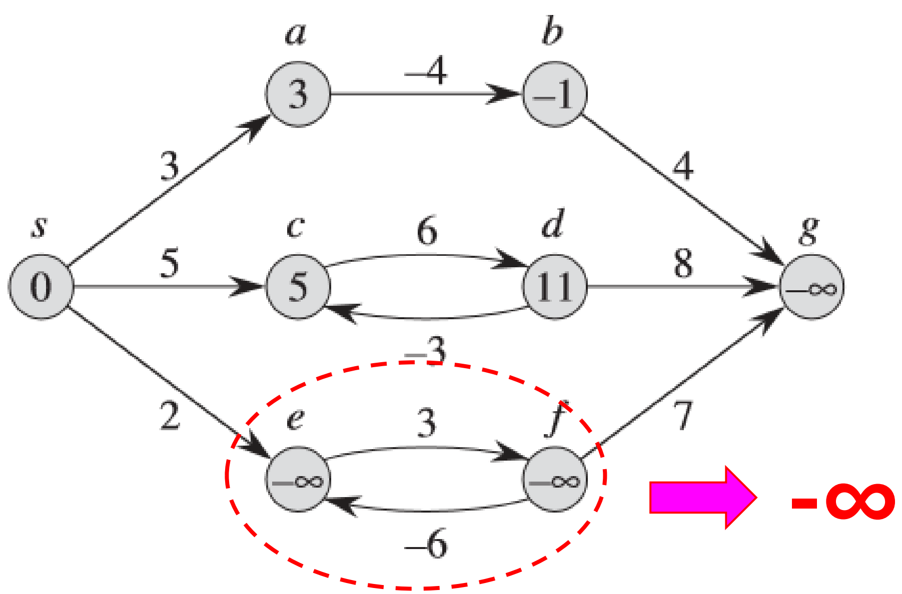
“这样”的最短路径具有**最优子结构性**：即两个结点之间的**最短路径中的任何子路径**都是最短的。

引理24.1 给定一个带权重的有向图 $G = (V, E)$ 和权重函数 $w: E \rightarrow R$。设 $p = <v_0, v_1, \ldots, v_k>$ **为从结点 v_0 到结点 v_k 的一条最短路径**，并且对于任意的 i 和 j，$0 \le i \le j \le k$，设 $p_{ij} = <v_i, v_{i+1}, \ldots, v_j>$ **为路径 p 中从结点 v_i 到结点 v_j 的子路径**，则 p_{ij} **是从结点 v_i 到结点 v_j 的一条最短路径。**（剪切-粘贴法证明，略，见P375）

3、负权重的边

权重为负值的边称为**负权重的边**。

如果存在负权重的边，则有可能存在**权重为负值的环路**，而造成图中**最短路径无定义**（此时可在负权重环路中“**转圈**”而使得路径的权重为 -∞）。



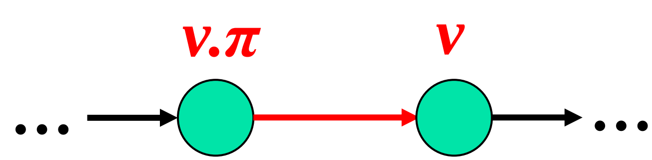
4、简单路径

- 如果一条路径中包含**两个相同的结点**，则该路径包含**环路**。
- **不包含环路的路径**称为**简单路径**。
- **最短路应为简单路径**，不包含环路。
  - 对任何简单路径，最多包含 $|V|$ **个结点和** $|V| - 1$**条边**。

不失一般性，假设后续算法寻找的最短路径都不包含环路。

5、前驱子图

记一个结点的**前驱结点**为：$v.\pi$



- 一个结点的前驱结点或者为 **NIL**（空，没有前驱结点）或者为另一个结点。

前驱子图：一个由源点 s 所诱导的**前驱子图**定义为 $G_\pi=(V_\pi, E_\pi)$，其中，

- **结点集合** $V_\pi = \{ v \in V：v.\pi \neq NIL \} \cup \{s\}$

  即 V_π 是源点 s 和图 G 中所有**有前驱结点的结点**的集合。

- **边集合** $E_\pi=\{ (v.\pi, v) \in E：v \in V_\pi-\{s\} \}$

  即 E_π 是由 V_π 中的结点 v 的 π **值所“诱导”**(induced)的边的集合，即**“前驱边”**构成的集合。

G_π 性质：单源点最短路径算法终止时，**G_π 是一棵最短路径树**。

该树包含了**从源点 s 到 s 可达的每个结点的最短路径**。

——**连通、无环图：树**。

一棵**根结点为 s 的最短路径树**是一个有向子图 $G'=(V', E')$，这里 $V' \subseteq V, E' \subseteq E$。且有以下性质：

（1）以 s 为根结点。

（2）V' 是图 G 中从源结点 s 出发可以到达的所有结点的集合（包括源点 s）。

（3）对于任意结点 $v \in V'$，图 G' 中**从结点 s 到结点 v 的唯一简单路径**是图 G 中**从结点 s 到结点 v 的一条最短路径**。

只有 G 不包含从 s 可以到达的权重为负值的环路时，最短路径才有定义，才有根结点为 s 的最短路径树。

5、松弛操作（Relax）

对于每个结点 ***v***，维持一个属性 ***v.d***，记录**从源点 *s* 到结点 *v* 的最短路径权重的上界**，称 ***v.d*** 为 ***s*** 到 ***v*** 的**最短路径估计**。

以下过程**INITIALIZE-SINGLE-SOURCE**对每个结点的最短路径估计 ***d*** 和前驱结点 ***π*** 进行初始化：

```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ G.V
        v.d = ∞
        v.π = NIL
s.d = 0
```

初始化后，对所有的结点 ***v***∈***V*** 有，

- ***v.π*** = ***NIL***;
- 源点 ***s*** 有 ***s.d*** = **0**;
- 而其它结点 ***v*** (***v***∈***V***-{***s***})有：

  v.d = ∞。

INITIALIZE-SINGLE-SOURCE 的时间： $\Theta(V)$

松弛操作： 首先测试一下有没有从 *s* 经过另一个结点 *u* 和边 (*u*, *v*) 而到达 *v* 的更短的路径。如果有，则 *v.d* 更新为新的最短路径估计值，同时 *v* 的 *v.π* 更新为 *u*，作为它的新的前驱结点。

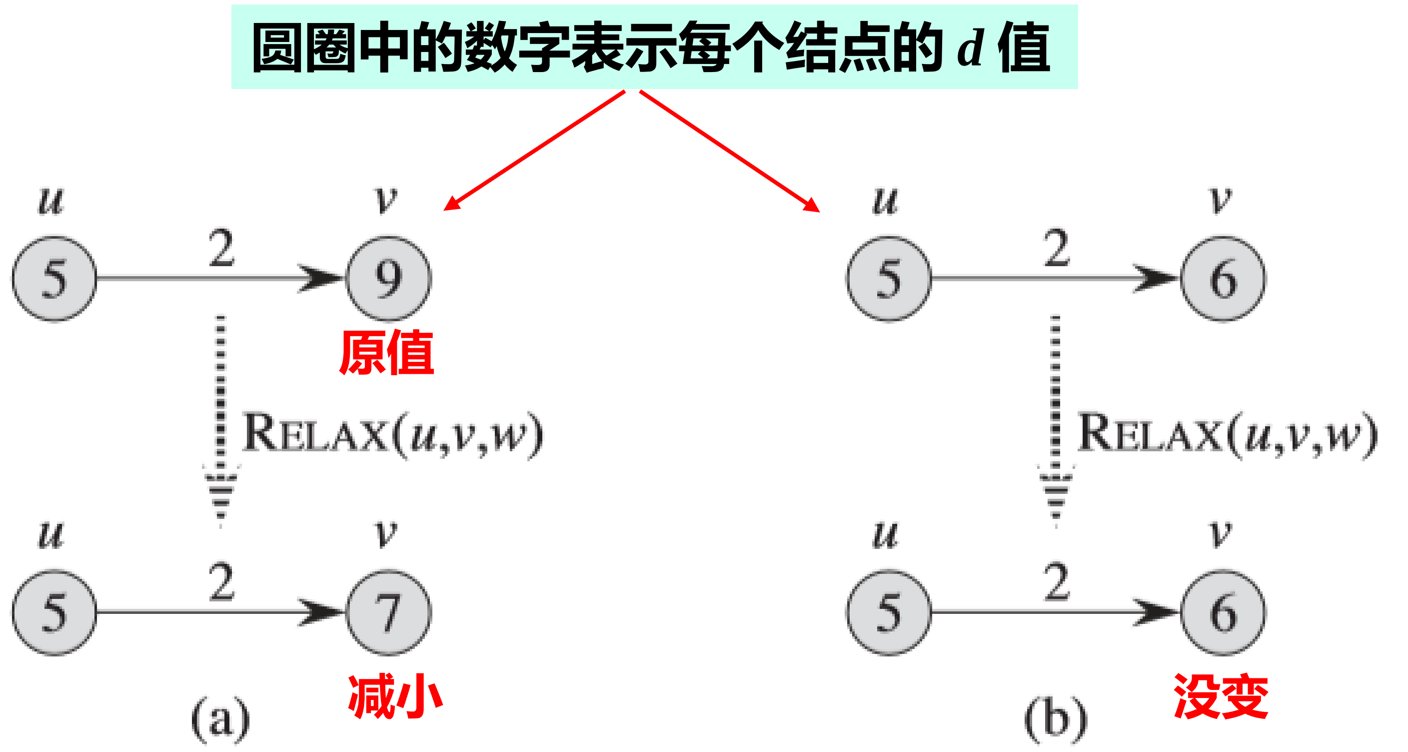
```
RELAX(u, v, w)
    if v.d > u.d + w(u, v)
        v.d = u.d + w(u, v)
        v.π = u
```

含义：根据最短路径的最优子结构，从 *s* 出发、经过结点 *u* 及边 (*u*, *v*) 到达结点 *v* 的**最短路径权重估计**等于*u.d* + *w*(*u*, *v*)。如果路径权重估计小于当前的 *v.d*，则代表有**一条更短的从 *s* 到达 *v* 的路径**，故更新 *v.d* 和 *v.π*。

从 *s* 到 *u* 的最短路径权重估计是 *u.d*

RELAX 的时间：*O*(1)

例：



对边 (u, v) 进行松弛操作，权重 $w(u, v) = 2$：

(a) 因为 $v.d > u.d + w(u, v)$，所以 $v.d$ 的值**减小**了，同时 $v.\pi = u$。

(b) 因为 $v.d \le u.d + w(u, v)$，所以 $v.d$ 的值没变，$v.\pi$ 也没有变化。

6、最短路径和松弛操作的相关性质

（1）三角不等式性质

引理24.11 设 $G = (V, E)$ 为一个带权重的有向图，其权重函数为 $w:E\rightarrow R$，设源结点为 s，那么对于所有的边 $(u, v)\in E$，有

$$\delta(s,v) \le \delta(s,u) + w(u,v)$$

证明：

假定 p 是从源结点 s 到结点 v 的一条最短路径，则 p **的权重**不会比任何从 s 到 v 的其它路径的权重大，因此路径 p 的权重也不会比这样的一条路径的权重更大：

从源结点 s 到结点 u 的最短路径 + 边 (u, v) 而构成的 s 到 v 路径。

而如果 s 到 v 没有最短路径，则不可能存在 s 到 v 的路径。 ■

(2) 上界性质： $v.d$ 是 s 到 v 的最短路径权重 $\delta(s, v)$ 的**上界**。

引理24.11 设 $G = (V, E)$ 为一个带权重的有向图，其权重函数为 $w: E \rightarrow R$，设源结点为 s，且该图由算法 INITIALIZE-SINGLE-SOURCE(G, s) 执行了初始化。那么对于所有的结点 $v \in V$，$v.d \geq \delta(s, v)$。并且**该不变式在对图 G 的边进行任何次序的松弛过程中都保持成立**，而且**一旦 $v.d$ 取得其下界 $\delta(s, v)$ 后，将不再发生变化**。

用**数学归纳法**证明：

基础步：INITIALIZE-SINGLE-SOURCE(G, s) 中进行初始化时，对于所有的结点 $v \in V - \{s\}$，置 $v.d = \infty$，而 $s.d = 0$，显然 $s.d \geq \delta(s, s)$，而其它的结点 $v.d \geq \delta(s, v)$，结论成立。

归纳步：考虑对边 (u, v) 的松弛操作。

假设在对边 (u, v) 进行松弛之前，对所有的结点 $x \in V$，都有

$$x.d \geq \delta(s, x)。$$

在**对边 (u, v) 进行松弛**时，唯一可能发生改变的 d 值只有 $v.d$，而如果该值发生变化，则有：

$$\begin{aligned} v.d &= u.d + w(u, v) \\ &\geq \delta(s, u) + w(u, v) && \text{(by the inductive hypothesis)} \\ &\geq \delta(s, v) && \text{(by the triangle inequality)}\ , \end{aligned}$$

同时，根据 $v.d$ 的定义和计算的规则，在 **$v.d$ 达到下界 $\delta(s, v)$** 后，就无法再减小（也不可能增加）。

引理得证。

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

(3) 非路径性质

推论24.12 给定一个带权重的有向图 $G=(V,E)$，其权重函数为 $w:E\rightarrow R$。假定**从源结点 s 到给定点 v 之间不存在路径**，则该图在由算法 *INITIALIZE-SINGLE-SOURCE*(G, s) 进行初始化后，有

$$v.d \geq \delta(s,v)=\infty,$$

并且该等式作为不变式一直维持到图 G 的所有松弛操作结束。

证明：

因为**从源点 s 到给定点 v 之间不存在路径**，所以 $\delta(s, v)=\infty$。而根据**上界性质**，总有 $v.d \geq \delta(s, v)$，所以，$v.d \geq \delta(s, v)=\infty$。

得证。

引理24.13 设 $G = (V, E)$ 是一个带权重的有向图，其权重函数为 $w: E \rightarrow R$，并且边 $(u, v) \in E$。那么在对边 (u, v) 进行松弛操作 RELAX(u, v, w) 后，有 $v.d \leq u.d + w(u, v)$ 。

证明：

如果在对边 (u, v) 进行**松弛操作前**，有 $v.d > u.d + w(u, v)$，则发生松弛时，置 $v.d = u.d + w(u, v)$。

如果在松弛操作前有 $v.d \leq u.d + w(u, v)$，则松弛操作并不会改变 $v.d$ 的值，因此在松弛操作后仍有 $v.d \leq u.d + w(u, v)$。

得证。

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

(4) 收敛性质

引理24.14 设 $G=(V,E)$ 为一个带权重的有向图，其权重函数为 $w:E\rightarrow R$。设 $s\in V$ 为某个源结点，$s \rightsquigarrow u \rightarrow v$ 为图 G 中**从 s 到 v 的一条最短路径**（从 s 经过 u 而到达 v，$u,v\in V$）。

假定图 G 由算法*INITIALIZE-SINGLE-SOURCE*(G, s) 进行初始化，并在这之后进行了**一系列边的松弛操作**，其中包括对边 (u,v) 的松弛操作 **RELAX**(u, v, w)。

如果在对 (u,v) 进行松弛操作之前的某时刻有 $u.d=\delta(s,u)$，则在该**松弛操作之后的所有时刻**有 $v.d=\delta(s,v)$。

即：算法最后能收敛于找到 s 到 v 的最短路径的状态（**至少能够找到从 s 经过 u 而到达 v 的最短路径**）。

证明：

根据**上界性质**有：如果在对边 **(*u*, *v*)** 进行松弛前的某个时刻有 $u.d = \delta(s, u)$，则**该等式在任何松弛之后仍然成立** (一直保持)。

那么在对边 **(*u*, *v*)** 进行松弛时将有，

$$
\begin{aligned}
v.d &\leq u.d + w(u, v) && \text{(by Lemma 24.13)} \\
&= \delta(s, u) + w(u, v) \\
&= \delta(s, v) && \text{(by Lemma 24.1)} .
\end{aligned}
$$

最短路的最优子结构性

而根据上界性质又有 $v.d \geq \delta(s, v)$。所以有 $v.d = \delta(s, v)$，根据**上界性质**，该等式在此之后一直保持成立。

得证。

(5) 路径松弛性质

引理24.15 设 $G = (V, E)$ 为一个带权重的有向图，其权重函数为 $w:E\rightarrow R$。设 $s\in V$ 为某个源结点，考虑**从源结点 s 到结点 v_k 的任意一条最短路径** $p = \langle v_0, v_1, v_2, \ldots, v_k\rangle$，$v_0 = s$。

如果图 G 由算法 *INITIALIZE-SINGLE-SOURCE*(G, s) 进行初始化，并在这之后进行了一系列边的松弛操作，其中包括对边 (v_0, v_1)、(v_1, v_2)、…、(v_{k-1}, v_k) 按照**所列次序**而进行的一系列松弛操作，则在所有这些松弛操作之后，有 $v_k.d = \delta(s, v_k)$，并且在此之后该等式一直保持。

并且该性质的成立与其他边的松弛操作及次序无关，即使这些松弛操作是与对 p 上的边所进行的松弛操作穿插进行的。

用归纳法证明：**从 $0 \sim k$，在对第 i 条边松弛之后有 $v_i.d = \delta(s, v_i)$。**

基础步：从初始化算法可以得出：$v_0.d = s.d = \delta(s, s)$，所以在对路径 p 的任何一条边进行松弛操作之前 ($i = 0$) 结论成立。且 $s.d$ 的取值在此之后不再发生变化。

归纳步：假定依次经过 (v_0, v_1)、(v_1, v_2)、…、(v_{i-2}, v_{i-1}) 松弛操作之后有 $v_{i-1}.d = \delta(s, v_{i-1})$，即**对 $v_1 \sim v_{i-1}$ 都找到了最短路**。

则在对边 (v_{i-1}, v_i) 进行松弛操时，根据**收敛性质**，松弛后必有 $v_i.d = \delta(s, v_i)$，并且该等式在此之后一直保持成立。

得证。

24.1 Bellman-ford算法

Bellman-ford算法可以求解**一般情况下**的单源点最短路径问题——**可以有负权重的边，但不能有负权重的环**。

设 $G = (V, E)$ 为一个带权重的有向图，权重函数为 $w:E \to R$。$s \in V$ 为源结点。

```
BELLMAN-FORD(G, w, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    for i = 1 to |G.V| - 1
        for each edge (u, v) ∈ G.E
            RELAX(u, v, w)
    for each edge (u, v) ∈ G.E
        if v.d > u.d + w(u, v)
            return FALSE
    return TRUE
```

Bellman-ford算法通过对所有的边反复进行松弛操作来**渐近地降低从源点 s 到每个结点 v 的最短路径的估计值 $v.d$**，直到该估计值与实际的最短路径权重 $\delta(s, v)$ 相同时为止。

算法返回TRUE当且仅当图 G 中不包含从源结点可达的权重为负值的环路

24.1 Bellman-ford算法

Bellman-ford算法可以求解一般情况下的单源最短路径问题

—— 可以有负权重的边，但不能有负权重的环

设 $G = (V, E)$ 为一个带 … $\omega: E \rightarrow R$。$s \in V$为源结点。

```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

Bellman-ford算法通过对所有的边 … 路径权重δ(s,v)相同的为止。

$v.d = \delta(s,v)$

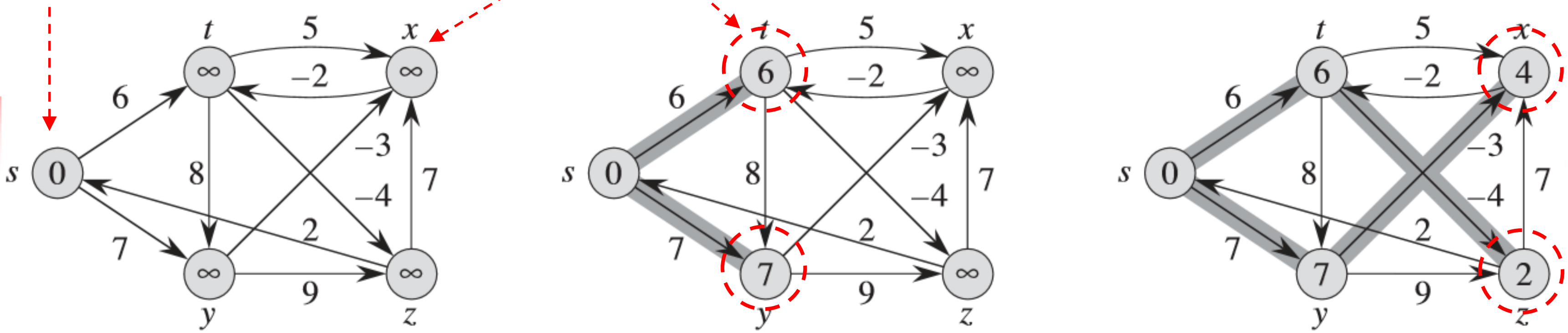
$v.\pi = u$

算法返回TRUE当且仅当图G中不包含 … 源结点可达的权重为负值的环路。

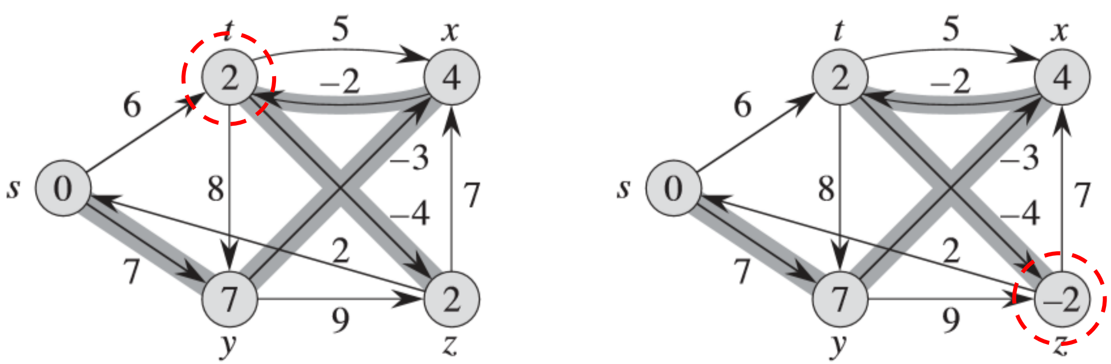
源结点：*s*

结点中的数值是结点的 ***d*** 值

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```



(a) 初始化后 (b) 第一次松弛操作后 (c) 第二次松弛操作后



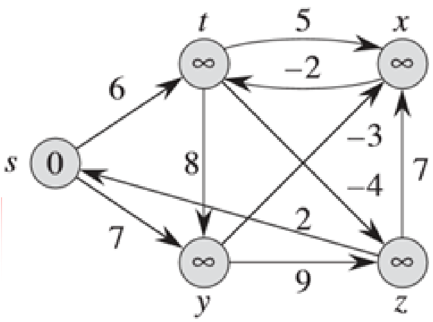
(d) 第三次松弛操作后 (e) 第四次松弛操作后

初始化后，*for* 循环将执行 |*V*|-1 次，每次对所有的边进行一次松弛处理。因此算法对图中每条边进行了|*V*| - 1 **次松弛处理**。

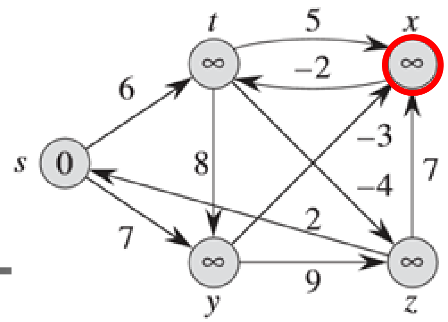
松弛边的顺序：$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$

- **加了阴影的边表示前驱值**：如果边 **(*u*, *v*)** 加了阴影，则 ***v.π*** **=** ***u***。
- 本例中**Bellman-ford**算法执行**4**次松弛操作后返回 **TRUE**。

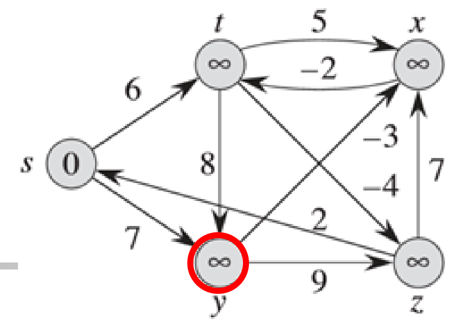
第一次松弛的过程 $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$



a）初始化后

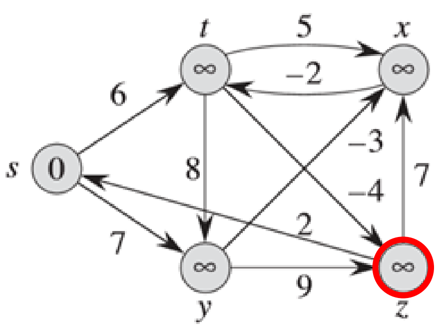


b.1) 对边(t,x)进行松弛

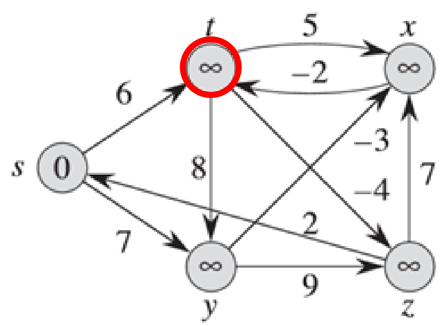


b.2) 对边(t,y)进行松弛

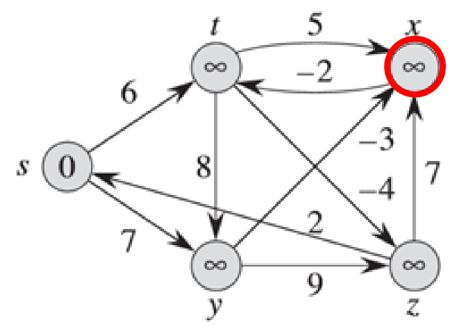
RELAX(u, v, w)
1 **if** $v.d > u.d + w(u, v)$
2 $\quad v.d = u.d + w(u, v)$
3 $\quad v.\pi = u$



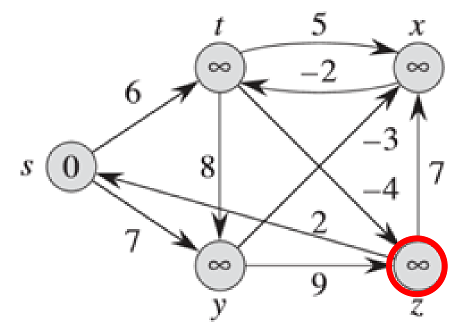
b.3) 对边(t,z)进行松弛



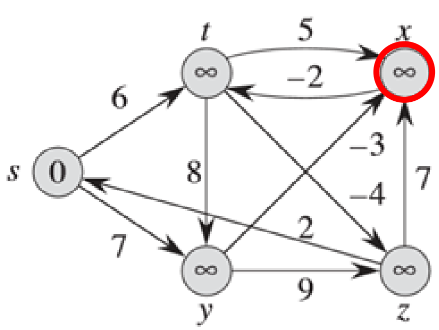
b.4) 对边(x,t)进行松弛



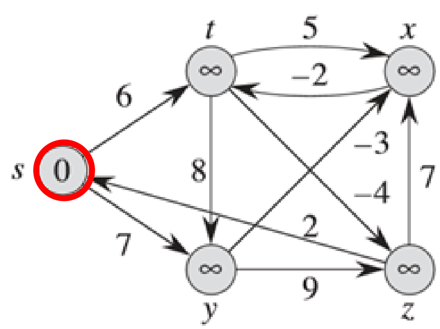
b.5) 对边(y,x)进行松弛



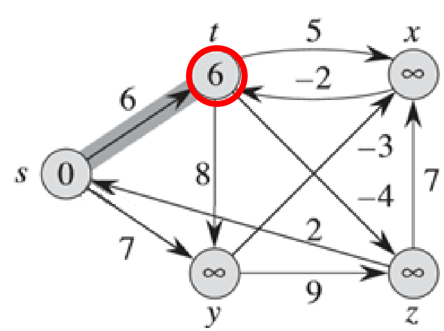
b.6) 对边(y,z)进行松弛



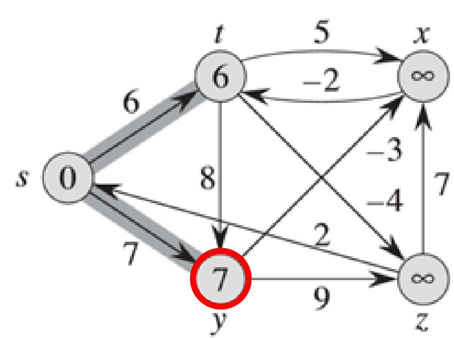
b.7) 对边(z,x)进行松弛



b.8) 对边(z,s)进行松弛



b.9) 对边(s,t)进行松弛

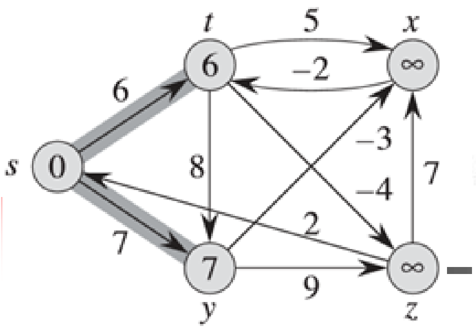


b.10) 对边(s,y)进行松弛

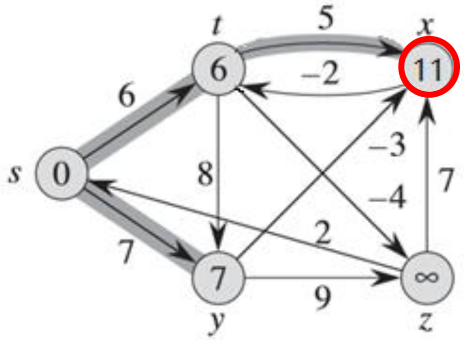
- 加了阴影的边表示前驱值：如果边（u, v）加了阴影，则 $v.\boldsymbol{\pi} = u$.

第二次松弛的过程

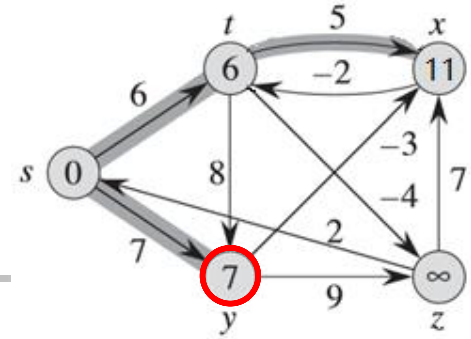
$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$



b) 第一次松弛操作后

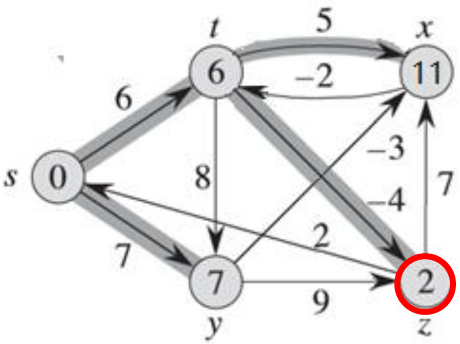


c.1) 对边(t,x)进行松弛

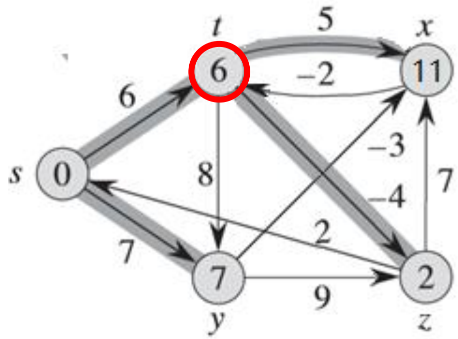


c.2) 对边(t,y)进行松弛

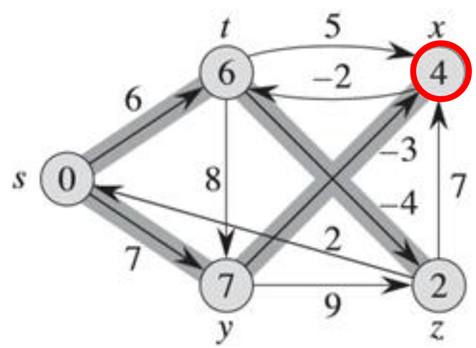
```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```



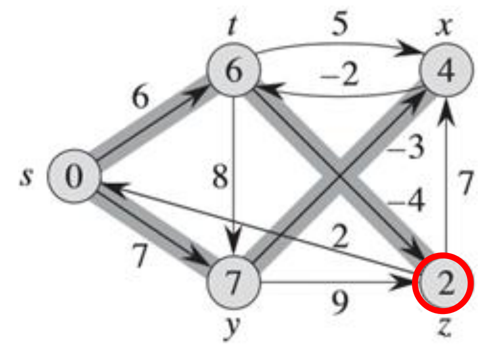
c.3) 对边(t,z)进行松弛



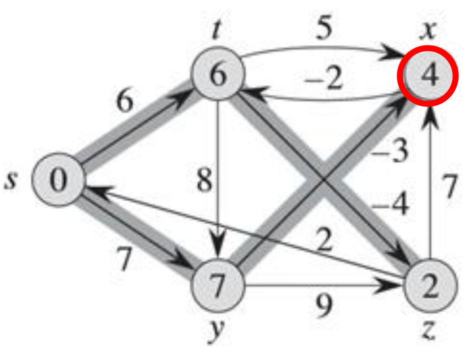
c.4) 对边(x,t)进行松弛



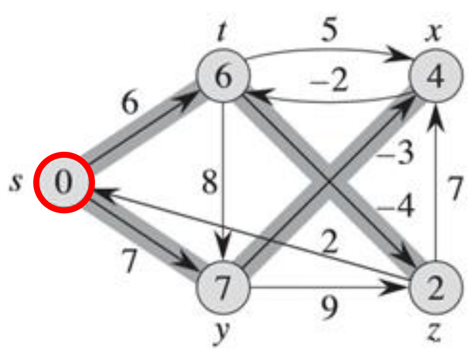
c.5) 对边(y,x)进行松弛



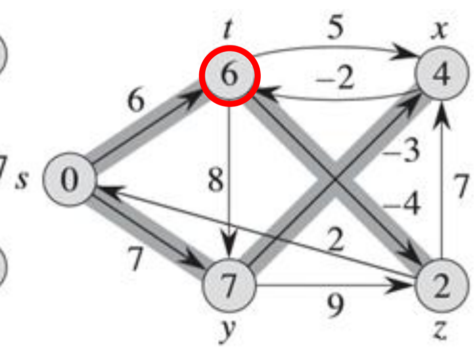
c.6) 对边(y,z)进行松弛



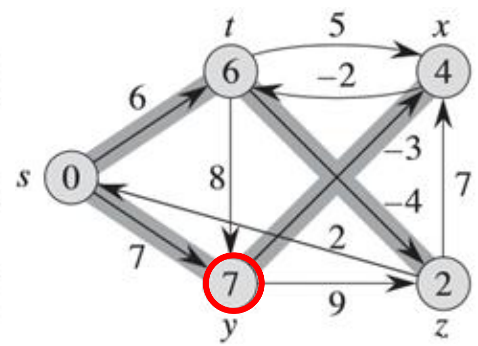
c.7) 对边(z,x)进行松弛



c.8) 对边(z,s)进行松弛



c.9) 对边(s,t)进行松弛



c.10) 对边(s,y)进行松弛

- 加了阴影的边表示前驱值：如果边 (u, v) 加了阴影，则 $v.\pi = u$.

◆ Bellman-ford算法的运行时间

初始化：$\Theta(V)$

松弛处理：*for* 循环执行 |*V*| -1 次，

每次循环是 Θ (*E*)

BELLMAN-FORD(G, w, s)

```
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

O(E)

- Bellman-ford算法总的运行时间是 *O* (*VE*)。

INITIALIZE-SINGLE-SOURCE(G, s)

```
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

$\Theta(V)$

RELAX(u, v, w)

```
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

$\Theta(1)$

◆ Bellman-ford算法是一个**动态规划**算法：

for all u $v.d = \min(v.d,\ u.d + w(u, v))$

Bellman-ford算法的证明

设 $G = (V, E)$ 为一个带权重的源点为 s 的有向图，其权重函数为 $w: E \rightarrow R$，并假定图 G 中不包含从源结点 s 可以到达的权重为负值的环路。

引理24.2 **Bellman-ford算法的第2~4行的** ***for*** **循环在执行了 $|V| - 1$ 次后，对于所有从源结点 s 可以到达的结点 v 有 $v.d = \delta(s, v)$.**

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| - 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

v.d 到达下界

证明：(使用**路径松弛性质**证明)

考虑任意从源结点 s 可以到达的结点 v。设 $p = <v_0, v_1, \ldots, v_k>$ **是从结点 s 到结点 v 的任意一条最短路径**，这里 $v_0 = s$， $v_k = v$。

因为最短路径是简单路径，所以 p 中最多包含 $|V| - 1$ 条边，故 $k \le |V| - 1$。

而对于路径 p，一定是**先求出 v_0 到最近的结点 v_1 的最短子路径**，再依次由近及远地求出到 v_2、$v_3\ldots$，直到 v_k 的最短子路径。所以对各条边松弛时将严格按照 (v_0, v_1)、(v_1, v_2)、…、(v_{k-1}, v_k) 的顺序进行的。

而算法第 2 ~ 4 行的 *for* 循环每次都对所有的 $|E|$ 条边进行松弛，所以也包含对上述各条边的松弛，那么根据**路径松弛性质**，在执行 $|V| - 1$ 次循环之后必有：$v.d = v_k.d = \delta(s, v_k) = \delta(s, v)$。得证.

引理24.3 对所有结点 $v \in V$，存在一条从源结点 s 到结点 v 的路径**当且仅当**Bellman-ford算法终止时有 $v.d < \infty$ 。

证明：（略）

定理24.4（Bellman-ford算法的正确性） 设Bellman-ford算法运行在一个带权重的源点为 s 的有向图 $G = (V, E)$上，其权重函数为 $w:E \rightarrow R$。

如果图 G 中**不包含从源结点 s 可以到达的权重为负值的环路**，则算法将返回 TRUE，且对于所有结点 $v \in V$，前驱子图 G_π 是一个根结点为 s 的最短路径树。

而如果图 G 中**包含一条从源结点 s 可以到达的权重为负值的环路**，则算法将返回FALSE。

定理24.4（Bellman-ford算法的正确性）的证明：

1) **首先证明**：如果图 G 中不包含从源结点 s 可以到达的权重为负值的环路，则算法将返回 TRUE，且对于所有结点 $v \in V$，前驱子图 G_π 是一个根结点为 s 的最短路径树。

证明：

(1) 证明：**对于所有结点 $v \in V$，在 $|V| - 1$ 次松弛结束时，有 $v.d = \delta(s, v)$**。

如果**结点 v 是从 s 可以到达**的，则论断可以从**引理24.2**得到证明。而如果**结点 v 不能从 s 达到**，则论断可以从**非路径性质**获得。

所以对于所有结点 $v \in V$，在 $|V| - 1$ **次松弛**结束时有 $v.d = \delta(s, v)$.

(2) 综合前驱子图性质和本论断，可以推导出 **G_π 是一棵最短路径树**。

图 G 中不包含从源结点 s 可以到达的权重为负值的环路时

(3) 终止时，算法是否返回 **TRUE**？

$|V|$ - 1 次松弛终止时，对所有的边 $(u, v) \in E$，有

$$
\begin{aligned}
v.d &= \delta(s, v) \\
&\le \delta(s, u) + w(u, v) \quad \text{(by the triangle inequality)} \\
&= u.d + w(u, v) \,,
\end{aligned}
$$

而 G 中不包含从源结点 s 可以到达的权重为负值的环路。因此，算法第6行中**没有任何测试**可以让算法返回 **FALSE**，故**最后一定返回TRUE值**。

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| - 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

2）然后证明：如果图 G 中包含一条从源结点 s 可以到达的权重为负值的环路，则算法将返回 **FALSE**。

证明：

假定图 G 包含一个权重为负值的环路，并且该环路可以从源结点 s 到达。设该环路为 $c = \langle v_0, v_1, \ldots, v_k \rangle$，这里 $v_0 = v_k$。

因为环路的权重为负值，所以有：

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0 \qquad (24.1)$$

反证法证明:

假设此种情况下Bellman-ford算法还是返回TRUE值。

由于对所有的 $i = 1, 2, \ldots, k$ 有

$$v_i.d \le v_{i-1}.d + w(v_{i-1}, v_i)$$

现在将**环路** c 上的所有这种不等式都加起来，有：

$$\begin{aligned}\sum_{i=1}^{k} v_i.d &\le \sum_{i=1}^{k}(v_{i-1}.d + w(v_{i-1}, v_i)) \\ &= \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)\end{aligned}$$

由于 $v_0 = v_k$，环路 c 上面的每个结点在 $\sum_{i=1}^{k} v_i.d$ 和 $\sum_{i=1}^{k} v_{i-1}.d$ 中都刚好各出现一次。因此有:

$$\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d$$

$$\boxed{\sum_{i=1}^{k} v_i.d} \le \sum_{i=1}^{k}(v_{i-1}.d + w(v_{i-1}, v_i))$$
$$= \boxed{\sum_{i=1}^{k} v_{i-1}.d} + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

进而有：$0 \le \sum_{i=1}^{k} w(v_{i-1}, v_i)$

这与 $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$ 矛盾（负权重的环路，边的权值之和应小于0）。

综上所述，如果图 G 中不包含从源结点 s 可以到达的权重为负值的环路，则算法将返回 **TRUE**，否则返回 **FALSE**。　得证。

24.3 Dijkstra算法

Dijkstra算法也是求解带权重有向图单源最短路径问题的算法，但该算法要求**图中不能有负权重的边和环**，**所有边的权重必须为非负值**。但该算法是贪心算法，速度块。

对所有结点的 d 值和 π 值初始化，$s.d = 0$.

DIJKSTRA(G, w, s)

INITIALIZE-SINGLE-SOURCE(G, s)

$S = \emptyset$

$Q = G.V$

while $Q \neq \emptyset$

  $u =$ EXTRACT-MIN(Q)

  $S = S \cup \{u\}$

  for each vertex $v \in G.Adj[u]$

    RELAX(u, v, w)

算法维护一个结点集合 S，集合中的每个结点都已经求出从 s 到该结点的最短路径。

对 Q 初始化

Q 是一个**最小优先队列**，保存结点集 V-S，**按结点的 d 值排序**。算法每次从 Q 中选择当前最短路径估计(d)最小的结点 u，将 u 从 Q 中删除，并加入到 S 中，$u.d$ 就是源结点s 到 u 的最短路径的长度。

仅对所有从 u 出发的边进行松弛。
然后重复上述过程，直到 $Q = \emptyset$。

Dijkstra算法是一个**贪心算法**：每次总是选择 $V - S$ 集合中**最短路径估计值最小**的结点加入 S 中。

```
DIJKSTRA(G, w, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    S = ∅
    Q = G.V
    while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        S = S ∪ {u}
        for each vertex v ∈ G.Adj[u]
            RELAX(u, v, w)
```

$s.d = 0$，s 是从 Q 中第一个被选中的结点

每个结点有且仅有一次机会从 Q 中被提取出来并加入 S 中。而一旦 u 被从 Q 中提取出来，$u.d$ 就是 s 到 u 的最短路径长度(且不再改变)。

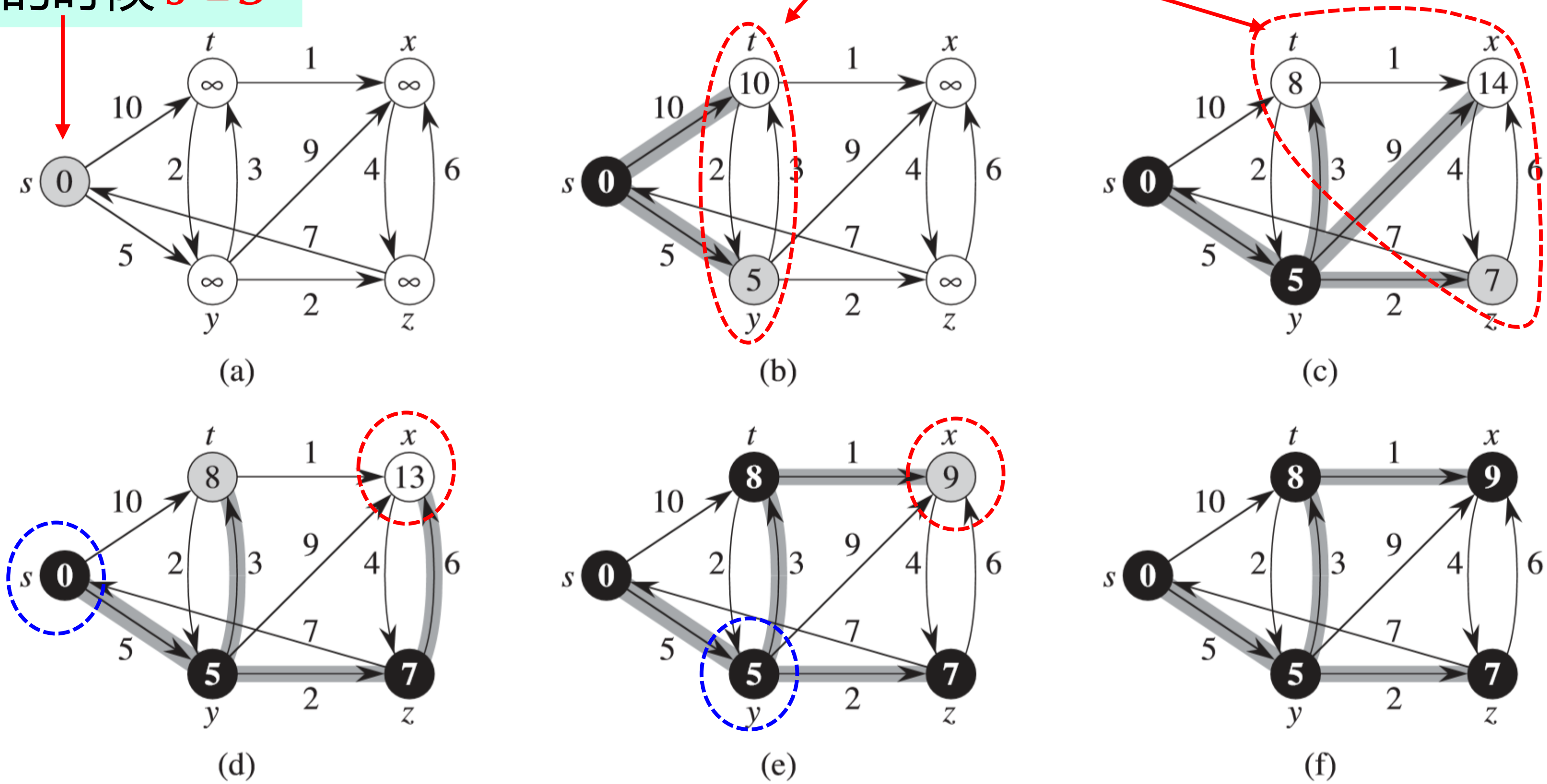
u 加入 S 后，**对从 u 出发的边 (u, v) 进行松弛**。而如果 $v.d$ 变小，则是因为存在一条**从 s 经过 u 到达 v 的更短路径**所致。此时，修改 $v.d = u.d + w(u, v)$，$v.\pi = u$，即 v 结点的新前驱为 u。

while 循环总共执行了 $|V|$ 次

源结点：s

开始的时候 $s \in S$

结点中的数值是 s 到该结点的最短路径的估计值

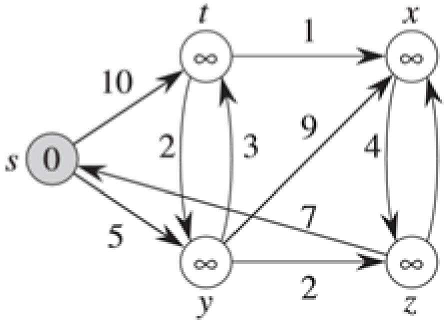


例， **Dijkstra算法**的执行过程

- 加了阴影的边表明前驱值（当前 u 出发的边）。
- 黑色的结点属于 S，白色的结点属于 $V - S$。加阴影的结点是算法下一次循环将选择加入 S 的点。

(a) **初始状态** $S = \{\ \}$

$Q = \{s, t, x, y, z\}$



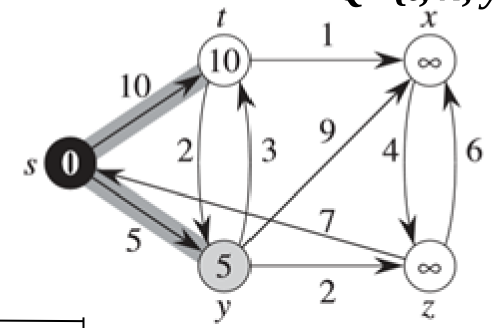
(a)

在 Q 中选择的结点 s

对边 (s, t)、(s, y) 松弛

(b)**首次选择后** $S = \{ s \}$

$Q=\{t, x, y, z\}$



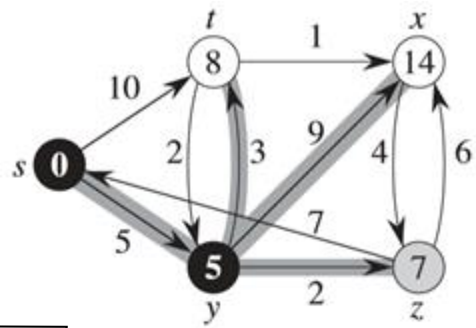
(b)

(c)**第二次选择后** $S = \{s, y\}$

$Q = \{t, x, z\}$

在 Q 中选择的结点 y

对边 (y, t)、(y, x)、(y, z)松弛



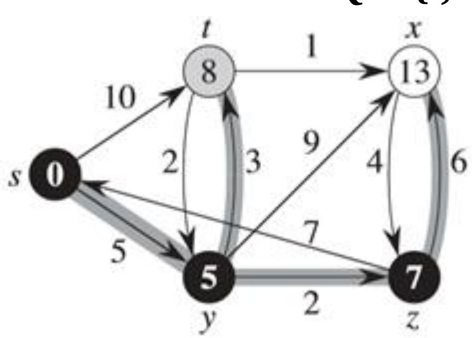
(c)

在 Q 中选择的结点 z

对边 (z, s)、(z, t)松弛

(d)**第三次选择后** $S = \{s, y, z\}$

$Q = \{t, x\}$



(d)

在 Q 中选择的结点 t

对边 (t, y)、(t, x)松弛

……

- 加了阴影的边表明前驱值(当前 u 出发的)。
- 黑色的结点属于 S，白色的结点属于V-S。加阴影的结点是算法下一次循环将选择加入 S 的点

定理24.6（**Dijkstra算法**的正确性）设**Dijkstra算法**运行在带权重的有向图 $G = (V, E)$ 上。如果所有边的权重为非负值，则在算法终止时，对于所有结点 $u \in V$，有 $u.d = \delta(s, u)$。

证明：（**利用循环不变式证明**）

循环不变式：算法在 *while* 语句的每次循环开始前，对于 S 中的每个结点 u 有 $u.d = \delta(s, u)$。

现在只需证明：对于每个结点 $u \in V$，当 u 被加入到 S 时，有 $u.d = \delta(s, u)$。因为一旦 u 加入 S，就不再修正 $u.d$，根据上界性质，该等式将一直保持。

证明过程：

(1) 初始化：初始时，$S = \varnothing$，因此循环不变式直接成立。

(2) 保　持：(在每次循环中，对于加入到集合 S 中的结点 u 而言，

$u.d = \delta(s, u)$)。

反证法证明：

设结点 u 是上述算法计算过程中出现的**第一个**在加入到集合 S 时 $u.d \neq \delta(s, u)$ **的结点**，之前的结点 k 都有 $k.d = \delta(s, k)$，并都加入了 S。

由于 s 是第一个加入 S中的结点，并且 $s.d = \delta(s, s) = 0$，所以可以合理假设 $u \neq s$ 并且 u 即将加入 S 时有 $S \neq \varnothing$（因为 S 中至少包含了 s）。

另外可以**合理假设** $u.d$!= ∞。

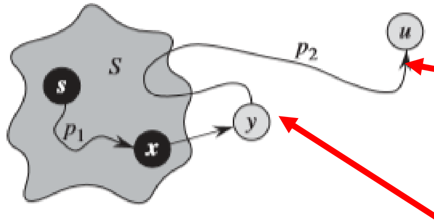
(因为根据算法规则，$u.d$ 是目前 V-S中最小的权值估计，若 $u.d = \infty$ 则表示 s 不可达 u，也就意味着 V-S 中的其它所有结点都不可达，算法可以终止了）。

同时，u 能加入集合 S 表示从 s 出发，经过之前的某个结点 k 可以到达 u，则对 u 而言，**必存在至少一条从 s 到 u 的路径**，而这样也意味着**一定存在从 s 到 u 的最短路径**。

（只是不是现在找到的这条路径 —— 因为 $u.d \neq \delta(s, u)$，所以现在找到的这条路径不是最短路径）

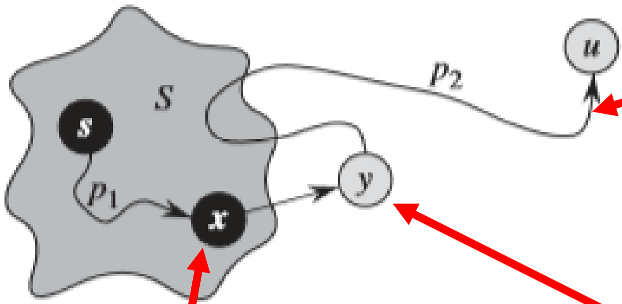
记**从 s 到 u 的最短路径**为 p。则 p上至少存在一个结点现在还没有被算法处理到。

因为根据假设处理到 u 之前的结点都有 $k.d = \delta(s, k)$，所以如果 p 中 u 之前的结点都被算法处理了，则它们都将有 $k.d = \delta(s, k)$，根据收敛性质，此时松弛到 u 的边就必有 $u.d = \delta(s, u)$，与此假设的情况矛盾，所以 p上至少存在一个结点现在还没有被算法处理.



一定存在 s 到 u 的最短路径 p

y 是 p上目前还没计入 S 的一个结点



在将结点 u 加入集合 S 之前，p 连接的是集合 S 中的结点 s 和 $V-S$ 中的结点 u。

考虑路径 p 上第一个满足 $y \in V - S$ 的结点 y，并设 y 的前驱是结点 x，$x \in S$，如图所示。路径分为三段：

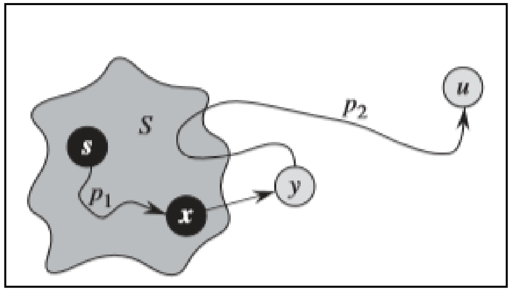
$$s \overset{p_1}{\leadsto} x \rightarrow y \overset{p_2}{\leadsto} u$$

注：x 有可能是 s 本身，y 也有可能是 u 本身（事实上也只能是 u 本身，除非 $\delta(y, u) = 0$）。

则：**在结点 u 加入到集合 S 时，应有 $y.d = \delta(s, y)$。**

这是因为 $x \in S$，u 是第一个 $u.d \neq \delta(s, u)$ 的结点，在将 x 加入到集合 S 时，有 $x.d = \delta(s, x)$，y 是 x 的邻接点，所以此时边 (x, y) 将被松弛。由于 y 是最短路径 p 上的结点，根据**最短路径的最优子结构性**和收敛性质，此时应有 $y.d = \delta(s, y)$。

因为结点 y 是从结点 s 到结点 u 的一条最短路径上位于 u 前面的一个结点，所以应有 $\delta(s, y) \le \delta(s, u)$，因此



$$y.d = \delta(s, y)$$

$$\le \delta(s, u)$$

$$\le u.d$$ （by the upper-bound property）

而在算法第5行选择结点 u 时，结点 u 和 y 都还在集合 $V - S$ 里，所以有 $u.d \le y.d$（思考为什么）。因此上述的不等式事实上只能是等式，即：$y.d = \delta(s, y) = \delta(s, u) = u.d$。

这与假设的 $u.d \ne \delta(s, u)$ 相矛盾，因此假设不成立。

所以，u在加入 S 时，将有 $u.d = \delta(s, u)$，该等式在随后的循环中一直保持。

终止：在算法终止时，$Q = \emptyset$，$S = V$。

根据前面**保持性的证明**，终止时对于所有的结点 $u \in V$，有 $u.d = \delta(s, u)$。

证毕。

推论24.7 如果在带权重的有向图 $G = (V, E)$ 上运行**Dijkstra算法**，其中的权重皆为非负值，源结点为 s，则在算法终止时，**前驱子图 G_π** 是一棵**根结点为 s 的最短路径树**。

从**定理24.6**和前驱子图性质可证（证明略）。

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

Dijkstra算法运行时间分析

（1）根据算法的处理规则，每个结点 u 仅被加入集合 S 一次，邻接链表 $Adj[u]$ 中的每条边在整个运行期间也只被检查一次。因此**算法第7-8行的 *for* 循环中，RELAX总共执行了 $|E|$ 次**（即松弛的总次数）。

（2）**Dijkstra算法**的总运行时间依赖于**最小优先队列 Q** 的实现：

如果用**线性数组** (无序或者按序插入) 实现，每次找 d 最小的结点 u 需要 $O(V)$ 的时间，所以算法的总运行时间为 $O(V^2 + E) = O(V^2)$。

如果用**二叉堆**实现，每次找 d 最小的结点 u 需要 $O(\lg V)$ 的时间，所以算法的总运行时间为 $O((V + E)\lg V)$。

如果用**斐波那契堆**实现，算法的总运行时间可改善至 $O(V\lg V + E)$

.

不管是**Bellman算法**还是**Dijkstra算法**，算法结束时（成功执行）都求出了每个结点的 $v.d$、$v.\pi$，并得到前驱子图 G_π。怎么利用 G_π 找出从 s 出发至各个结点的最短路径上的结点序列呢?

上述计算过程结束后，对每个结点 $v \in V$ 都求出了 $v.d$、$v.\pi$，利用 $v.\pi$ 反推，即可得到 s 至 v 路径上的结点序列。

24.4 差分约束系统和最短路径

线性规划问题：

给定一个 $\boldsymbol{m \times n}$ 的矩阵 $\boldsymbol{A}$、一个 $\boldsymbol{m}$ 维向量 $\boldsymbol{b}$ 和一个 n 维向量 $\boldsymbol{c}$，求一个 $\boldsymbol{n}$ 维向量 $\boldsymbol{x}$，使得在 $\boldsymbol{Ax \le b}$ 的约束下，使目标函数：

$$\sum_{i=1}^{n} c_i x_i$$

最大。

本节讨论线性规划的一个特例：**差分约束系统**。

差分约束系统中，矩阵 A 的每一行包括一个 **1** 和一个 **-1**，其它所有项都是为 **0**，由 $Ax \le b$ 给出的约束条件形式上是 m **个涉及** n **个变量的差额限制条件** (difference constraints)，每个约束条件是以下简单的**线性不等式**：

$$x_j - x_i \le b_k,$$ 这里 $1 \le i, j \le n$，$i \ne j$，并且 $1 \le k \le m$。

由这些**线性不等式**组成的**线性不等式组**称为**差分约束系统**。

例：求满足下列条件的

5 维向量 $x = (x_i)$：

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \leqslant \begin{bmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{bmatrix}$$

上述问题的一般形式：

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \leqslant \begin{bmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{bmatrix}$$

$$\begin{cases} x_1 - x_2 \le 0, \\ x_1 - x_5 \le -1, \\ x_2 - x_5 \le 1, \\ x_3 - x_1 \le 5, \\ x_4 - x_1 \le 4, \\ x_4 - x_3 \le -1, \\ x_5 - x_3 \le -3, \\ x_5 - x_4 \le -3. \end{cases}$$

线性不等式组

- 找一组满足上述约束条件的解。
- 问题可能的答案有：x = (-5, -3, 0, -1, -4)、

  x' = (0, 2, 5, 4, 1) 等，有**多组可行解**。

这些解之间的一个基本关系是：

引理24.8 设向量 $x = (x_1, x_2, \ldots, x_n)$ 为差分约束系统 $Ax \le b$的一个解，设 d 为任意常数，则 $x + d = (x_1 + d, x_2 + d, \ldots, x_n + d)$ 也是该差分约束系统的一个解。

证明：

根据约束条件，对每对 x_i和 x_j，$(x_i + d) - (x_j + d) = x_i - x_j$。因此若向量 x 满足 $Ax \le b$，则向量 $x + d$ 也满足 $Ax \le b$。 ■

差分约束系统的应用举例

末知变量 x_i 代表事件发生的时间，每个约束条件给出的是在两个时间点之间**必须间隔的最短时间**。

比如，设这些事件是产品装配过程中的步骤：

如果在时刻 x_1 使用一种需要两个小时才能风干的粘贴剂，则下一个步骤需要 2 个小时后等粘贴剂干了之后才能在时刻 x_2 安装部件。这样就有约束条件 $x_2 \geq x_1+2$，亦即 $x_1 - x_2 \leq -2$ 等。

求解差分约束系统：

约束图：对给定的差分约束系统 $Ax \le b$，其对应的约束图是一个带权重的**有向图** $G = (V, E)$，这里，

v_0 是一个虚设的结点，作为整个图的源点。

其余结点 v_i 每个对应一个未知变量 x_i， $i = 1, 2, \ldots, n$.

$$V = \{v_0, v_1, \cdots, v_n\}$$

$$E = \{(v_i, v_j) : x_j - x_i \le b_k \text{ 是一个约束条件}\}$$

$$\cup \{(v_0, v_1), (v_0, v_2), (v_0, v_3), \cdots, (v_0, v_n)\}$$

- 每个不等式对应一条边：若有 $x_j - x_i \le b_k$，则约束图中存在一条**从** v_i **指向** v_j 的有向边 $\langle v_i, v_j \rangle$；
- 同时存在从 v_0 到其它所有结点的边 $\langle v_0, v_i \rangle$， $i = 1, 2, \ldots, n$。

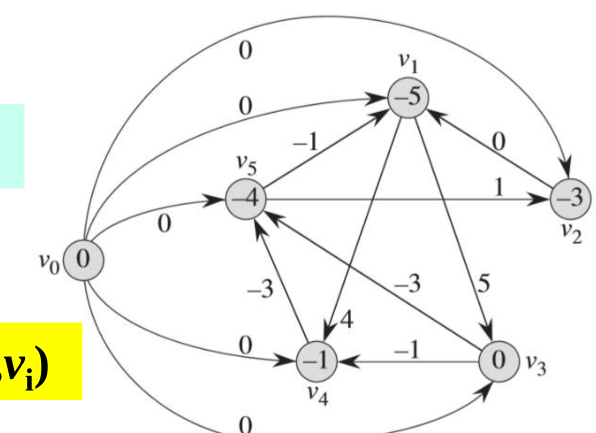
(1) **结点集合**：约束图中引入一个额外的结点 v_0，从其出发可以达到其它所有结点。**结点集合 V 由代表每个变量 x_i 的结点 v_i 和额外的结点 v_0 组成。**

(2) **边集合**：边集合 E 包含代表每个差分约束不等式的边，同时包含 v_0 到其它所有结点的边 (v_0, v_i)，$i = 1, 2, \ldots, n$。

(3) **边的权重**：如果 $x_j - x_i \le b_k$ 是一个差分约束条件，则边 (v_i, v_j) 的权重记为 $w(v_i, v_j) = b_k$，而从 v_0 出发到其它结点的边的权重 $w(v_0, v_j) = 0$。

上例的差分约束系统的约束图如下：

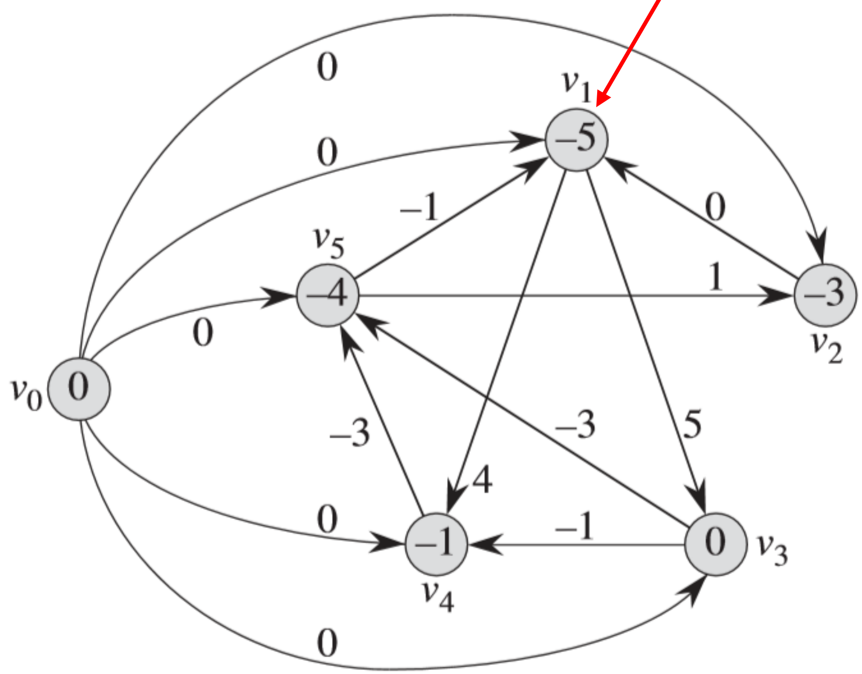
结点中的数值是 $\delta(v_0, v_i)$



$$
\begin{aligned}
x_1 - x_2 &\le 0, \\
x_1 - x_5 &\le -1, \\
x_2 - x_5 &\le 1, \\
x_3 - x_1 &\le 5, \\
x_4 - x_1 &\le 4, \\
x_4 - x_3 &\le -1, \\
x_5 - x_3 &\le -3, \\
x_5 - x_4 &\le -3.
\end{aligned}
$$

上例的差分约束系统的约束图如下：

结点中的数值是 $\delta(v_0, v_i)$



$$
\begin{aligned}
x_1 - x_2 &\le 0\,, \\
x_1 - x_5 &\le -1\,, \\
x_2 - x_5 &\le 1\,, \\
x_3 - x_1 &\le 5\,, \\
x_4 - x_1 &\le 4\,, \\
x_4 - x_3 &\le -1\,, \\
x_5 - x_3 &\le -3\,, \\
x_5 - x_4 &\le -3\,.
\end{aligned}
$$

- 结点集合 V 由代表每个变量 x_i 的结点 v_i 和额外的结点 v_0 组成
- 边集合 E 包含代表每个差分约束条件的边，同时包含 v_0 到其他所有结点的边 (v_0, v_i)。
- 边 (v_i, v_j) 的权重记为 $w(v_i, v_j) = b_k$， $w(v_0, v_j) = 0$。

定理24.9 给定差分约束系统 $Ax \le b$，设 $G = (V, E)$ 是该差分约束系统所对应的约束图。

(1) 如果图 G 不包含权重为负值的回路，则

$$x = (\delta(v_0, v_1), \delta(v_0, v_2), \delta(v_0, v_3), \ldots\ldots, \delta(v_0, v_n))$$

是该系统的一个可行解。

(2) 如果图 G 包含权重为负值的回路，则该系统没有可行解。

证明： 考虑任意一条边 $(v_i, v_j) \in E$，根据**三角不等式**有：

$$\delta(v_0, v_j) \le \delta(v_0, v_i) + w(v_i, v_j),$$

即： $\delta(v_0, v_j) - \delta(v_0, v_i) \le w(v_i, v_j)$

令：$x_i = \delta(v_0, v_i)$ 和 $x_j = \delta(v_0, v_j)$，则 x_i 和 x_j 就是满足差分约束条件 $x_j - x_i \le w(v_i, v_j)$ 的量。 $x_j - x_i \le b_k$ $w(v_i, v_j) = b_k$

因此，$x = (\delta(v_0, v_1), \delta(v_0, v_2), \delta(v_0, v_3), \ldots\ldots, \delta(v_0, v_n))$ 是问题的一个可行解。

（前提：**不包含权重为负的环路**）。

而如果约束图**包含权重为负值的环路**，不失一般性，设权重为负值的环路为 $c = \langle v_1, v_2, \ldots, v_k \rangle$，这里 $v_1 = v_k$。

环路 c 对应下面的差分约束条件组：

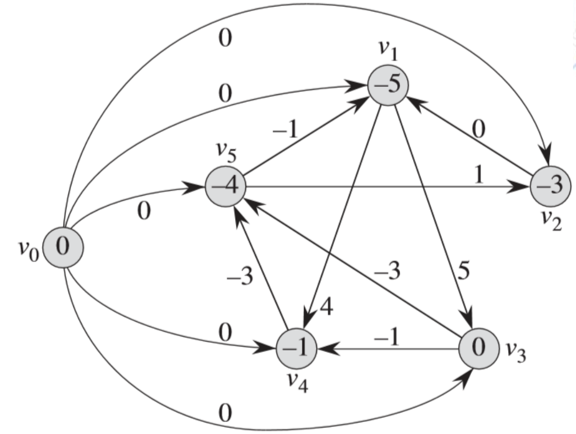
$$
\begin{aligned}
x_2 - x_1 &\leq w(v_1, v_2)\,, \\
x_3 - x_2 &\leq w(v_2, v_3)\,, \\
&\vdots \\
x_{k-1} - x_{k-2} &\leq w(v_{k-2}, v_{k-1})\,, \\
x_k - x_{k-1} &\leq w(v_{k-1}, v_k)\,.
\end{aligned}
$$

- 不等式左侧求和，等于 **0**。($v_1 = v_k$ ，所有 v_i 相互抵消)
- 不等式右侧求和，等于环路 c 的权重 $w(c)$，且有： $0 \leq w(c)$

—— **这与 c 是权重为负值的环路相矛盾。故该组不等式无解。**

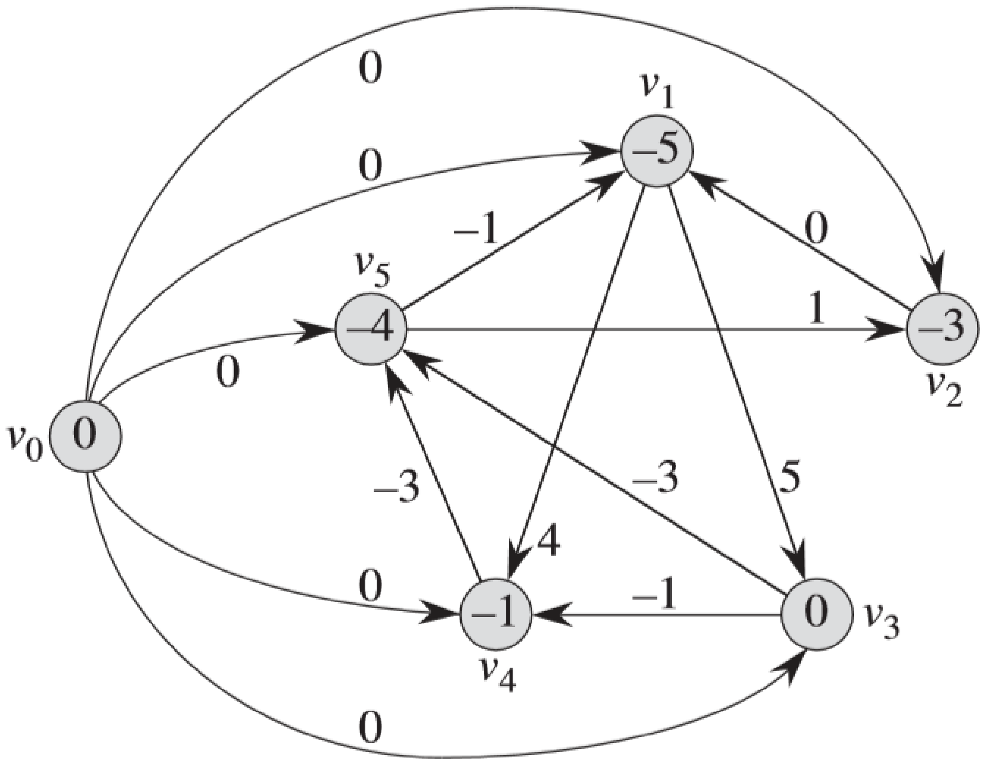
求解差分约束系统：

求解差分约束系统就是**求约束图中 v_0 到其它各结点的最短路径**，而因为约束图中有负权值的边，所以要用 **Bellman-Ford 算法**。



- 如果Bellman-Ford**算法**返回TRUE，则最短路径权重 $\delta(v_0, v_i)$，$i = 1, 2, \ldots, n$，给出该系统的一个可行解。
- 如果算法返回FALSE，则该系统无解。

注：因为约束图中含有从源结点 v_0 到其他所有结点的边，所以若图中存在权重为负值的环路，则一定可以从 v_0 可达。



- 结点中的数值是 $\delta(v_0, v_i)$；
- 最短路径权重提供的一个可行解是：$x = (-5, -3, 0, -1, -4)$;
- 同时根据**引理24.8**，对于任意常数 d，$(d-5, d-3, d, d-1, d-4)$ 也是问题的解。