算法设计与分析

Computer Algorithm Design & Analysis

2020.10

群名称：算法设计与分析2020
群　号：271069522

吕志鹏
zhipeng.lv@hust.edu.cn

群名称：算法设计与分析
群　号：　271069522

Chapter 24
Single-Source Shortest Paths

单源最短路径

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ω:E→R。图中一条路径p=<$v_0, v_1, \ldots, v_k$>的权重ω(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δ(u,v)定义如下：

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

单源最短路径问题：

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

▪ 最短路径的最优子结构

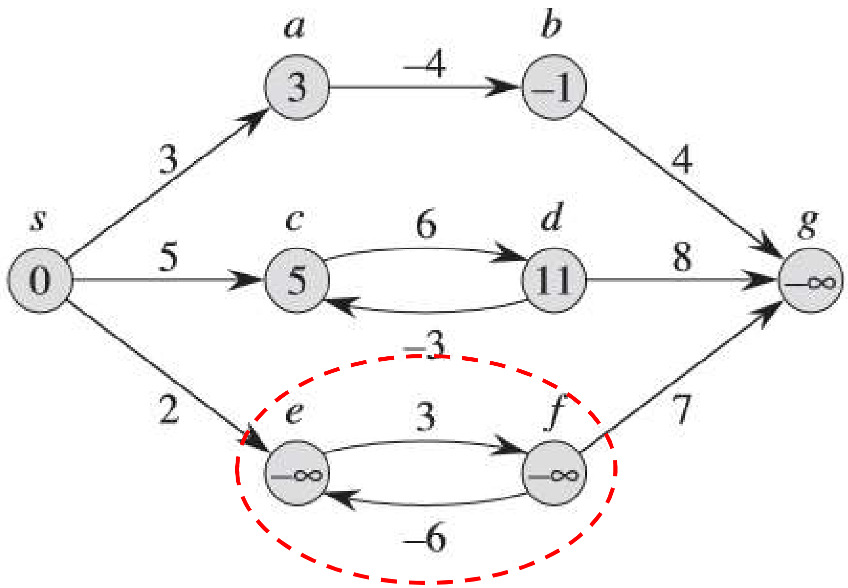
这样最短路径具有最优子结构性：两个结点之间的最短路径的任何子路径都是最短的。

引理24.1 给定一个带权重的有向图G=(V,E)和权重函数$\omega: E\rightarrow R$。设$p=\langle v_0, v_1, \cdots, v_k\rangle$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，$0\leqslant i\leqslant j\leqslant k$，设$p_{ij}=\langle v_i, v_{i+1}, \cdots, v_j\rangle$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。（证明略，见P375）

■ 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（路径的权重为-∞）。



- 环路
  - 最短路不应包含环路。
    - 不包含环路的路径称为简单路径。
      - 对任何简单路径最多包含|V|-1条边和|V|个结点。
      - 不失一般性，假设后续算法寻找的最短路径都不包含环路。
- 最短路径的表示
  - 一个结点的前驱结点记为：v.π
    - 前驱结点或者为NIL或者为另一个结点
  - 利用v.π的记录可以搜索出最短路径上的所有结点。

■ 前驱子图

定义**前驱子图**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是图G中的前驱结点不为NIL的结点的集合，再加上源点s。
- 边集合$E_\pi=\{(v.\pi,v)\in E：v\in V_\pi-\{s\}\}$
  - 即E_π是由V_π中的结点的π值所“诱导”(induced)的边的集合。

则，算法终止时，**G_π是一棵最短路径树**。

- 该树包含了从源结点s到每个可以从s到达的结点的一条最短路径。

设G=(V,E)是一条带权重的有向图，其权重函数为ω:E→R，假定G不包含从s可以到达的权重为负值的环路，因此，所有的最短路径都有定义。

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） V′是图G中从源结点s可以到达的所有结点的集合。

（2） G′形成一棵根结点为s的树。

（3）对于所有的结点v∈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。

■ 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**。

- 过程INITIALIZE-SINGLE-SOURCE实现对结点最短路径估计和前驱结点的**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;
- s.d = 0;
- 对所有的结点v∈V-{s}有：v.d = ∞。

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

松弛操作：首先测试一下是否可以对从s到v的最短路径进行改善（即有没有更短的路径）。**如果可以改善，则v.d更新为新的最短路径估计值，v的前驱v.π更新为新的前驱结点。**

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

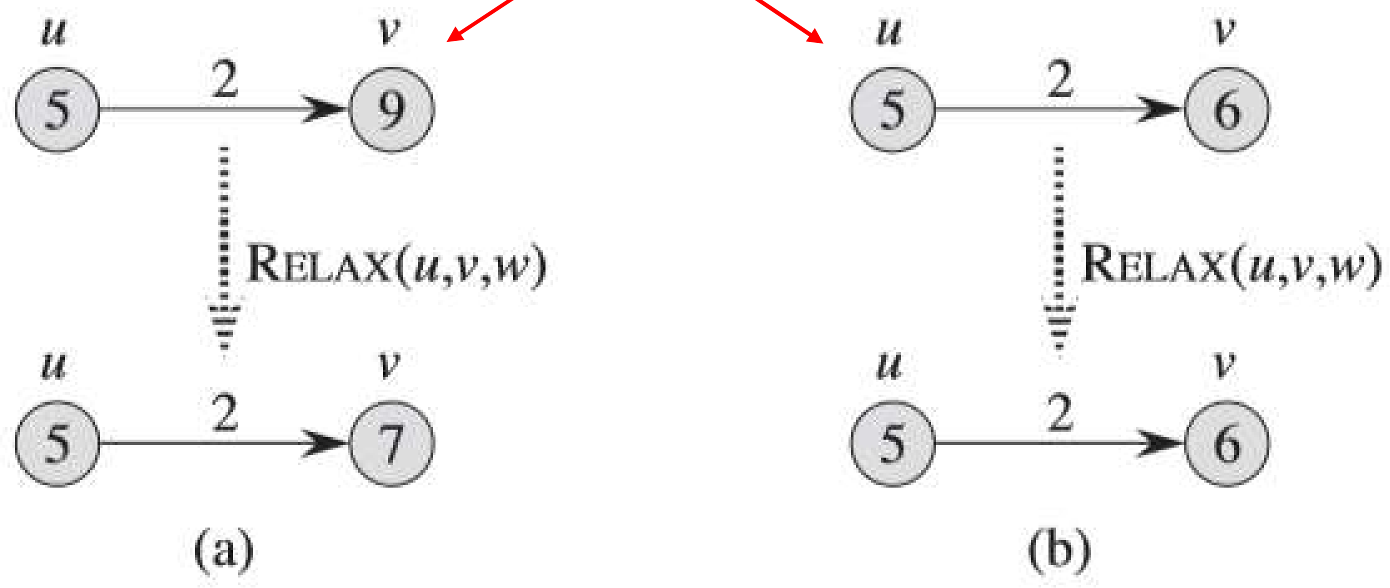
测试：对s到v所经过的**最后一个中间结点u**，按下列方式计算从s出发，经过u而到达v的路径的权重：

将从结点s到结点u之间最短路径加上结点u到v之间的边的权重，然后与当前的s到v的最短路径估计v.d进行比较，看有没有变小。如果变小，则对v.d和v.π进行更新——这一操作就称为“**松弛**”

RELAX 的时间：O(1)

例：

结点中的数字：每个结点的最短路径估计



对边(u,v)进行松弛操作，权重：ω(u,v)=2：

(a)：因为v.d > u.d + ω(u,v)，所以v.d的值减小(9 > 5+2)。

(b)：因为v.d ≤ u.d + ω(u,v)，所以v.d的值没有改变。

最短路径和松弛操作的性质

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ω:E→R，设其源结点为s。那么对于所有的边(u,v)∈E，有

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

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大：从源结点s到结点u的一条最短路径，再加上边(u,v)而到达结点v的这条路径。

如果s到v没有最短路径，则不可能存在到v的路径。 ■

2. 上界性质：v. d是s到v的最短路径权重δ(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ω:E→R，设其源结点为s，该图由算法 *INITIALIZE-SINGLE-SOURCE(G,s)* 执行初始化。那么对于所有的结点v∈V，v.d≥δ(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，而一旦v.d取得其下界δ(s,v)后，将不再发生变化。

用数学归纳法证明：对于所有的结点v∈V，v.d≥δ(s,v)。

- 注：归纳的主体是松弛步骤的数量。

基础步：在经 *INITIALIZE-SINGLE-SOURCE(G, s)* 初始化之后，对于所有的结点v∈V-{s}，置 v. d=∞ ，而s. d=0 ，显然s. d≥δ(s, s)，而其它的结点 v. d≥δ(s, v)，结论成立。

归纳步：考虑对边(u, v)的松弛操作。

归纳假设：在对边(u, v)进行松弛之前，对所有的结点x∈V，

x. d≥δ(s, x)。

而在对边(u, v)进行松弛的过程中，唯一可能发生改变的d值只有v. d，如果该值发生变化，则有：

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

同时，根据计算的规则，在v. d达到其下界δ(s, v)后，就无法再减小（也不可能增加）。

引理得证。

RELAX(u, ν, w)

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

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ω:E→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δ(s, v)=∞。而根据上界性质，总有v. d≥δ(s, v)，所以，v. d≥δ(s, v)=∞。

得证。

引理24.13 设G=(V,E)为一个带权重的有向图，其权重函数为ω:E→R，并且边(u,v)∈E。那么在对边(u,v)进行松弛操作RELAX(u,v,ω)后，有v.d≤ u.d+ω(u,v) 。

证明:

如果在对边(u, v)进行松弛操作前，有v. d>u. d+ω (u, v)，则松弛操作时，置v. d=u. d+ω (u, v)。

如果在松弛操作前有v. d≤ u. d+ω (u, v)，则松弛操作不会改变v. d和u. d的值，因此在松弛操作后仍有v. d≤u. d+ω (u, v).

得证。

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      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ω:E→R。设s∈V为某个源结点，$s \rightsquigarrow u \rightarrow v$ 为图G中的一条最短路径（u，v∈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,ω)。如果在对边(u,v)进行松弛操作之前的某时刻有u.d=δ(s,u)，则在该松弛操作之后的所有时刻有v.d=δ(s,v)。

证明:

根据上界性质，如果在对边(u, v)进行松弛前的某个时刻有u. d=δ(s, u)，则该等式在松弛之后仍然成立。

特别地，在对边(u, v)进行松弛后，有

$$
\begin{aligned}
v.d &\le u.d + w(u,v) && \text{(by Lemma 24.13)} \\
&= \delta(s,u) + w(u,v) \\
&= \delta(s,v) && \underset{\text{最优子结构性}}{\text{(by Lemma 24.1)}} \ .
\end{aligned}
$$

而根据上界性质，有v.d≥δ(s,v)。所以有v.d=δ(s,v)，并且该等式在此之后一直保持成立。

得证。

4. 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ω:E→R。设s∈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=<v_0,v_1,…,v_k>，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。

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。

归纳法证明：在最短路径p的第i条边被松弛之后，有$v_i.d=\delta(s,v_i)$

基础步：在对路径p的任何一条边进行松弛操作之前，从初始化算法可以得出：$v_0.d=s.d=\delta(s,s)$。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，$v_{i-1}.d=\delta(s,v_{i-1})$。

则在对边(v_{i-1},v_i)进行松弛操时，根据收敛性质，必有在对该边进行松弛后$v_i.d=\delta(s,v_i)$，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，其权重函数为ω:E→R。s∈V为源结点。

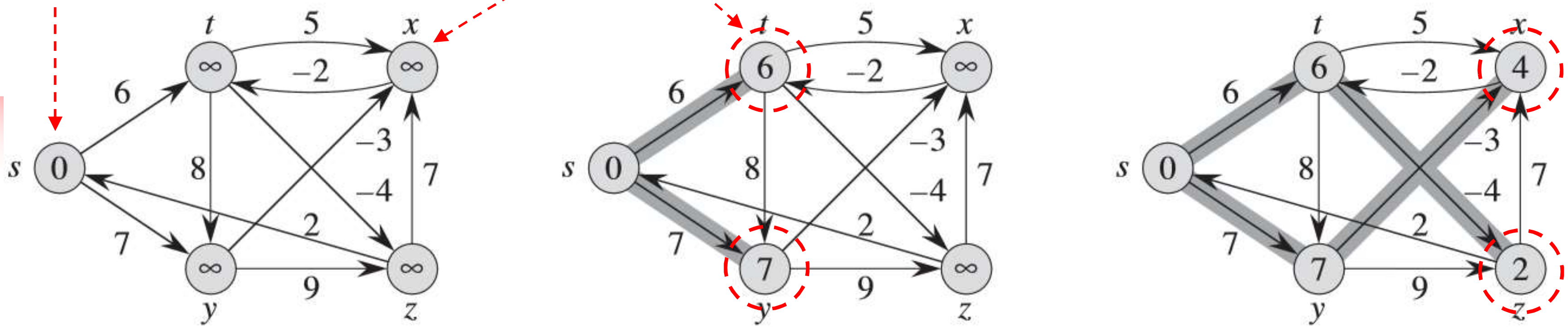
```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

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。

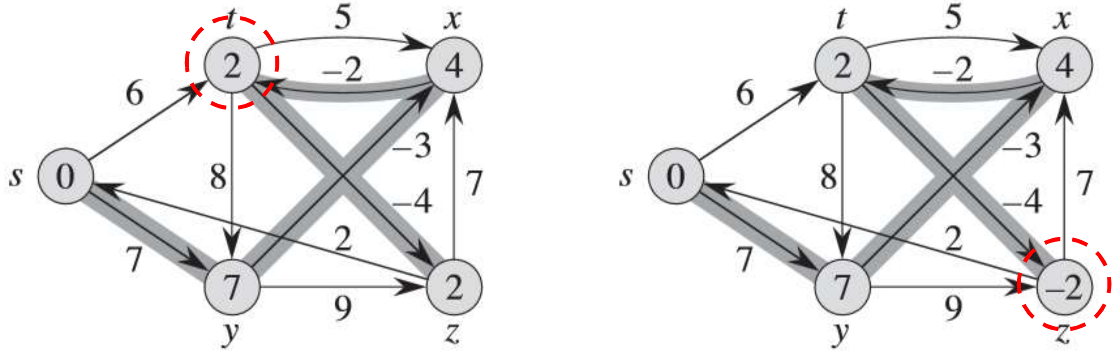
源结点：s

结点中的数值是结点的d值

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      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，算法将对图中每条边进行|V|-1次松弛处理：

for循环执行|V|-1次，每次对所有的边进行进行一次松弛处理。

例，Bellman-ford算法的执行过程

- 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。

Bellman-ford算法的运行时间

初始化：Θ(V)

松弛处理：for循环执行|V|-1次，

每次的时间是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O(VE)。

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

Θ(V)

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

Θ(1)

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ω:E→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|V|-1次之后，对于所有从源结点s可以到达的结点v有

v.d=δ(s,v)。

v.d到达下界

```
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
```

引理24.2 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。则Bellman-ford算法的第2~4行的for循环在执行|V|-1之后，对于所有从源结点s可以到达的结点v有v.d=δ(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
```

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, …, v_k⟩是从结点s到结点v之间的任意一条最短路径，这里v_0=s，v_k=v。

- 因为最短路径都是简单路径，所以p中最多包含|V|-1条边，故k≤|V|-1。
- 同时，算法第2~4行的for循环每次松弛所有的|E|条边，每一次为p最多扩展一条边（想想为什么？）。所以对序列⟨v_0, v_1, …, v_k⟩在其第i次松弛操作时，被松弛的边中包含边(v_{i-1}, v_i)，这里i=1, 2, …, k。
- 根据路径松弛性质有：v.d = v_k.d = δ(s, v_k) = δ(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$\omega:E \to R$。

- 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（略）：

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∈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∈V，在算法终止时，有v.d=δ（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。
- 如果结点v不能从s可达，则论断可以从非路径性质获得。

因此，对于所有结点v∈V，在算法终止时，有v.d=δ(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

（3）终止时，算法是否返回TRUE?

算法终止时，对所有的边(u,v)∈E，有

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

因此，算法第6行中没有任何测试可以让算法返回FALSE（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$c=<v_0, v_1, \ldots v_k>$，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算法返回TRUE值，

则有：对所有的i=1,2,…,k，

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

将环路c上的所有这种不等式都加起来，有：

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

由于$v_0=v_k$，环路c上面的每个结点在上述求和表达式

$\sum_{i=1}^{k} v_i.d$ 和$\sum_{i=1}^{k} v_{i-1}.d$ 中都刚好各出现一次。因此有

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

因此有 $$0 \le \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

与 $$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0 \, .$$ 相矛盾。

因此，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算法解决带权重的有向图上单源最短路径问题。
- **该算法要求所有边的权重均为非负值**，即对于所有的边(u,v)∈E，ω(u,v)≥0，—— 不能有负权重的边和环。

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

算法从结点集V-S中选择当前最短路径估计最小的结点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。这里Q是一个**最小优先队列**，保存结点集V-S。

然后对所有从u出发的边进行松弛。然后重复上述过程，直到Q=Ø。

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)
```

每个结点有且仅有一次机会被从Q中抽取并加入S中。一旦u被从Q中抽取出来，u.d就是s到u的最短路径长度（不再改变，上界性质）。

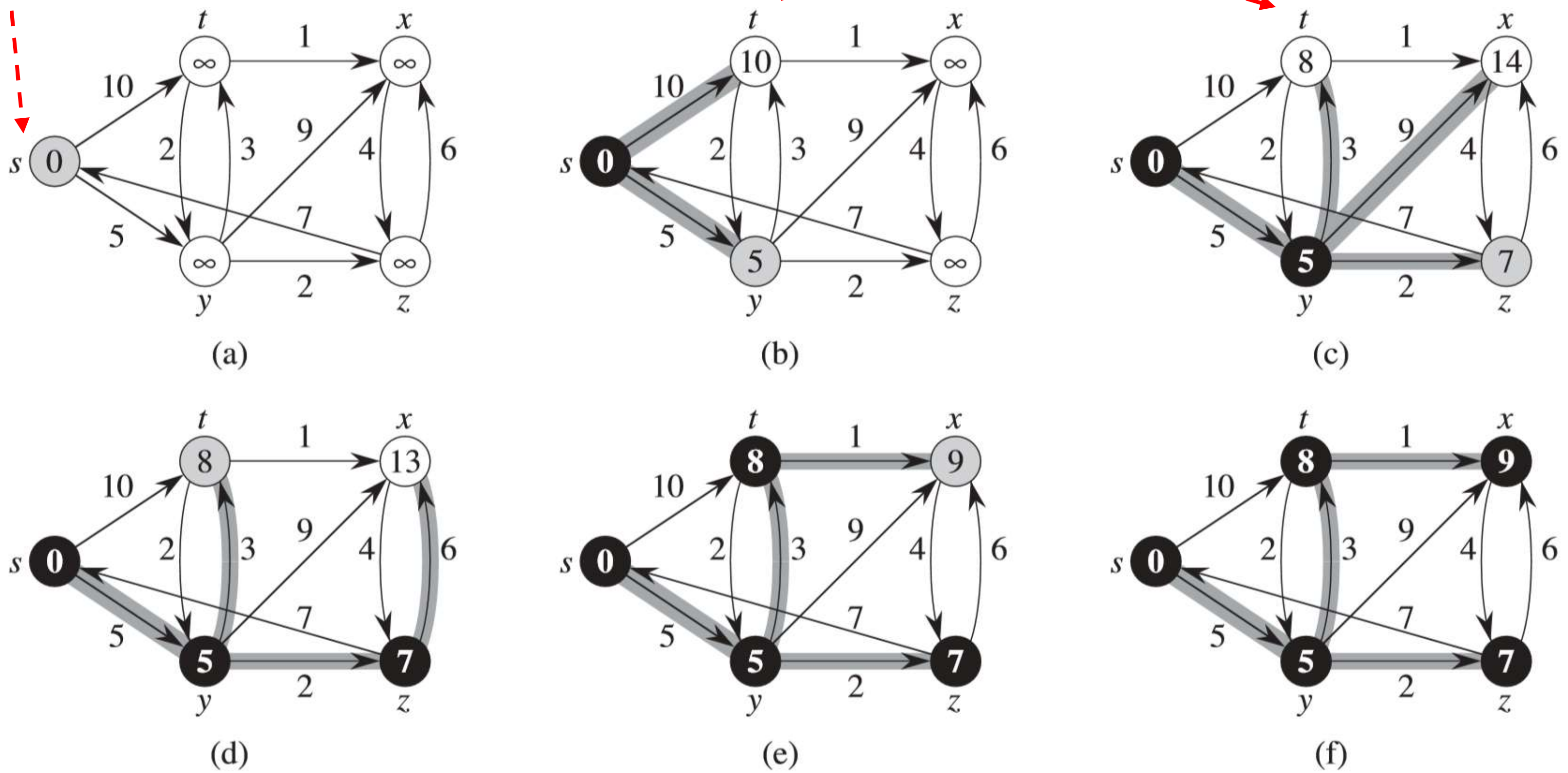
u加入S后，对从u出发的边的(u,v)进行松弛。而如果v.d变小，则是因为存在从s经过u到达v的更短路径所致。此时，修改v.d=u.d +ω(u,v)，v.π=u，即最短路径上v结点的新前驱为u。

while循环总共执行了|V|次

源结点：s

开始的时候s∈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。

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∈V，有u.d=δ(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，对于每个结点u∈S，有u.d=δ(s,u)

只需证明：对于每个结点u∈V，当u被加入到S时，有u.d=δ(s,u)。

注：一旦u加入S，就不会再修正u.d。且根据上界性质，该等式将一直保持。

证明过程：

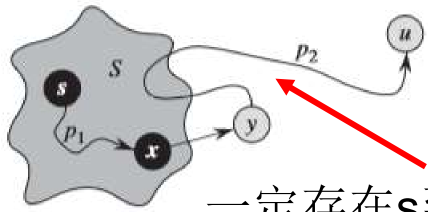
（1）初始化：初始时，$S=\varnothing$，因此循环不变式直接成立。

（2）保　持：在每次循环中，对于加入到集合S中的结点u而言，$u.d=\delta(s,u)$。

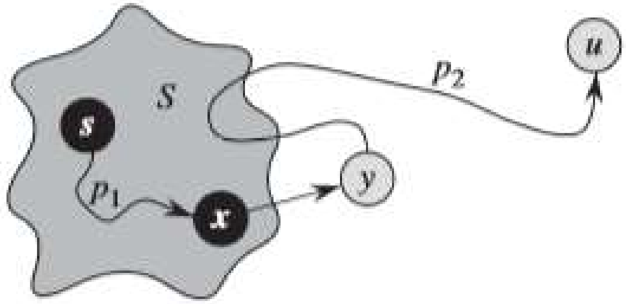
用反证法证明：**设结点u是第一个在加入到集合S时$u.d \neq \delta(s,u)$的结点。**

由于s是第一个加入到集合S中的结点，并且$s.d=\delta(s,s)=0$，所以$u \neq s$，并且在u即将加入S时，$S \neq \varnothing$，因为S中至少包含了s。

故，此时必存在至少一条从s到u的路径（否则，根据非路径性质将有$u.d=\delta(s,u)=\infty$，与假设的$u.d \neq \delta(s,u)$相矛盾，故这样路径一定存在），这样也必存在一条从s到u的**最短路径**，记为p。



一定存在s到u的最短路径p



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

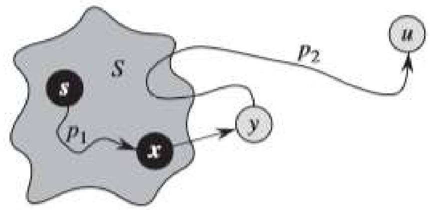
$$s \overset{p_1}{\leadsto} x \to y \overset{p_2}{\leadsto} u$$

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δ(y,u)=0）。

则有：**在结点u加入到集合S时，应有y.d=δ(s,y)。**

- 这是因为x∈S，u是第一个u.d≠δ(s,u)的结点，在将x加入到集合S时，有x.d=δ(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=δ(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δ(s,y)≤ δ(s,u)，因此



$$
\begin{aligned}
y.d &= \delta(s, y) \\
&\le \delta(s, u) \\
&\le u.d \qquad \text{(by the upper-bound property)}
\end{aligned}
$$

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≤y.d（思考为什么）。因此上式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≠δ(s,u)相矛盾。因此假设不成立。所以，u在加入S时，将有u.d=δ(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=Ø，S=V。

根据前面保持性的证明，终止时对于所有的结点u∈V，有u.d=δ(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

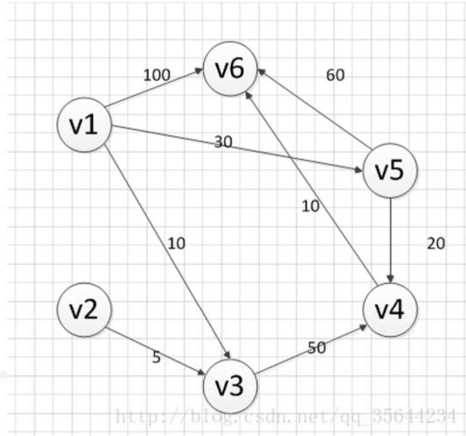
- 根据算法的处理规则，每个结点u仅被加入集合S一次，邻接链表Adj[u]中的每条边在整个运行期间也只被检查一次。因此**算法第7-8行的for循环执行次数总共为|E|次**（即松弛判定总次数）。

- 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以改善至$O(V\lg V+E)$。

SPFA算法介绍

- 算法思路：
  - 我们用数组dis记录每个结点的最短路径估计值，用邻接表或邻接矩阵来存储图G。是对Bellman-Ford算法的改进。
  - 采取的方法是动态逼近法：设立一个先进先出的队列用来保存待优化的结点，优化时每次取出队首结点u，并且用u点当前的最短路径估计值对离开u点所指向的结点v进行松弛操作.
  - 如果v点的最短路径估计值有所调整，且v点不在当前的队列中，就将v点放入队尾。这样不断从队列中取出结点来进行松弛操作，直至队列空为止.

- **初始化：**
- 首先我们先初始化数组dis如下图所示：（除了起点赋值为0外，其他顶点的对应的dis的值都赋予无穷大，这样有利于后续的松弛）



dis数组

v1	v2	v3	v4	v5	v6
0	∞	∞	∞	∞	∞

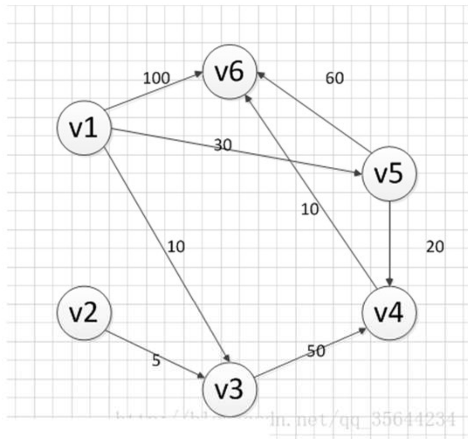
- **第一次循环：**
- 首先，队首元素出队列，即是v1出队列，然后，对以v1为弧尾的边对应的弧头顶点进行松弛操作，可以发现v1到v3，v5，v6三个顶点的最短路径变短了，更新dis数组的值，得到如下结果：

dis数组

v1	v2	v3	v4	v5	v6
0	∞	10	∞	30	100

- 我们发现v3，v5，v6都被松弛了，而且不在队列中，所以要他们都加入到队列中：**{v3，v5，v6}**

- **第二次循环**
- 此时，队首元素为v3，v3出队列，然后，对以v3为弧尾的边对应的弧头顶点进行松弛操作，可以发现v1到v4的边，经过v3松弛变短了，所以更新dis数组：



dis数组

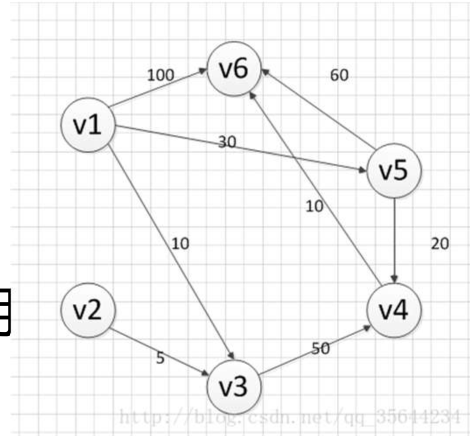
v1	v2	v3	v4	v5	v6
0	∞	10	60	30	100

- 此时只有v4对应的值被更新了，而且v4不在队列中，则把它加入到队列中：**{v5,v6,v4}**
- **第三次循环**
- 此时，队首元素为v5，v5出队列，然后，对以v5为弧尾的边对应的弧头顶点进行松弛操作，发现v1到v4和v6的最短路径，经过v5的松弛都变短了，更新dis的数组：

dis数组

v1	v2	v3	v4	v5	v6
0	∞	10	50	30	90

- 我们发现v4、v6对应的值都被更新了，但是他们都在队列中了，所以不用对队列做任何操作。队列值为：**{v6，v4}**



- **第四次循环:**
- 队首元素为v6，v6出队列，然后，对以v6为弧尾的边对应的弧头顶点进行松弛操作，发现v6出度为0，所以我们不用对dis数组做任何操作，其结果和上图一样，队列同样不用做任何操作，它的值为：**{v4}**
- **第五次循环:**
- 队首元素为v4，v4出队列，然后，对以v4为弧尾的边对应的弧头顶点进行松弛操作，可以发现v1到v6的最短路径，经过v4松弛变短了，所以更新dis数组

dis数组

v1	v2	v3	v4	v5	v6
0	∞	10	50	30	60

- 因为我修改了v6对应的值，而且v6也不在队列中，所以我们把v6加入队列，**{v6}**
- 第六次循环此时，队首元素为v6，v6出队列，然后，对以v6为弧尾的边对应的弧头顶点进行松弛操作，发现v6出度为0，所以我们不用对dis数组做任何操作，其结果和上图一样，队列同样不用做任何操作。所以此时队列为空。

SPFA算法原理

- SPFA(Shortest Path Faster Algorithm) [图的存储方式为邻接表] 是Bellman-Ford算法的一种队列实现，减少了不必要的冗余计算。
- 算法大致流程是用一个队列来进行维护。 初始时将源结点加入队列。每次从队列中取出一个元素，并对所有与它相邻的点进行松弛，若某个相邻的点松弛成功，则将其入队。 直到队列为空时算法结束。
- 它可以在O(kE)的时间复杂度内求出源点到其他所有点的最短路径，可以处理负边。

SPFA算法原理

- SPFA 在形式上和BFS非常类似，不同的是BFS中一个点出了队列就不可能重新进入队列，但是SPFA中一个点可能在出队列之后再次被放入队列，也就是一个点改进过其它的点之后，过了一段时间可能本身被改进，于是再次用来改进其它的点，这样反复迭代下去。

- 判断有无负环：如果某个点进入队列的次数超过V次则存在负环（SPFA无法处理带负环的图）。

- SPFA算法是Bellman-ford算法的队列优化，比较常用。SPFA算法在负边权图上可以完全取代Bellman-ford算法，另外在稀疏图中也表现良好。但是在非负边权图中，为了避免最坏情况的出现，通常使用效率更加稳定的Dijkstra算法，以及它的使用堆优化的版本。通常的SPFA算法在一类网格图中的表现不尽如人意。不是很稳定，不如Dijkstra。

SPFA算法改进：

- SPFA算法有两个优化算法 SLF 和 LLL：
  - **SLF：Small Label First 策略**，设要加入的节点是j，队首元素为i，若dist(j)<dist(i)，则将j插入队首，否则插入队尾。
  - **LLL：Large Label Last 策略**，设队首元素为i，队列中所有dist值的平均值为x，若dist(i)>x则将i插入到队尾，查找下一元素，直到找到某一i使得dist(i)<=x，则将i出队进行松弛操作。
- SLF 可使速度提高 15 ~ 20%；SLF + LLL 可提高约 50%。

24.4 差分约束和最短路径（自学）

线性规划：给定一个m×n的矩阵A、一个m维的向量b和一个n维的向量c。试找一n维向量x，使得在Ax≤b的约束下，目标函数 $\sum_{i=1}^{n} c_i x_i$ 最大。

- 求解线性规划问题：单纯形法等
- 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：在一个差分约束系统中，线性规划矩阵A的每一行包括一个1和一个-1，其它所有项皆为0。由Ax≤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关系：

$$\boxed{x_j - x_i \le b_k}$$

这里1≤i，j≤n，i≠j，并且1≤k≤m。

例：一个满足下列条件的5维向量$x=(x_i)$的问题：

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

上述问题的一般形式：

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

- 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$x+d=(x_1+d,x_2+d,\ldots, x_n+d)$也是个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材料，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。

约束图：

在一个Ax≤b的差分约束系统中，将m×n的矩阵A看成是一张有n个结点和m条边构成的图的邻接矩阵的转置。

约束图定义如下：

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

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

$$E=\{(v_i, v_j): x_j - x_i \leqslant b_k \text{ 是一个约束条件}\}$$
$$\cup \{(v_0, v_1), (v_0, v_2), (v_0, v_3), \cdots, (v_0, v_n)\}$$

每条有向边对应一个不等式，(v_0,v_i)是从新增的v_0到其他所有结点的边

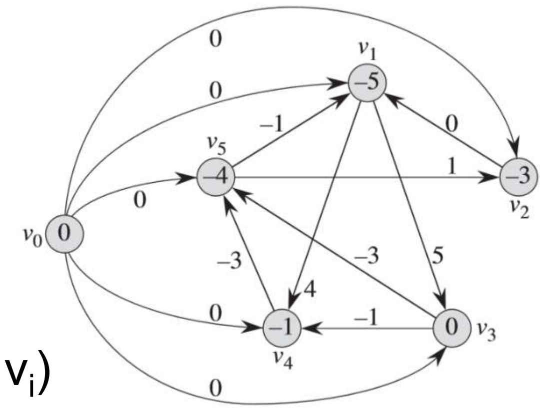
说明:

（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他所有结点。因此结点集合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的边，同时包含v_0到其他所有结点的边(v_0,v_i)，i=1,2，…，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$\omega(v_i,v_j) = b_k$，而从v_0出发到其他结点的边的权重$\omega(v_0,v_j)=0$。

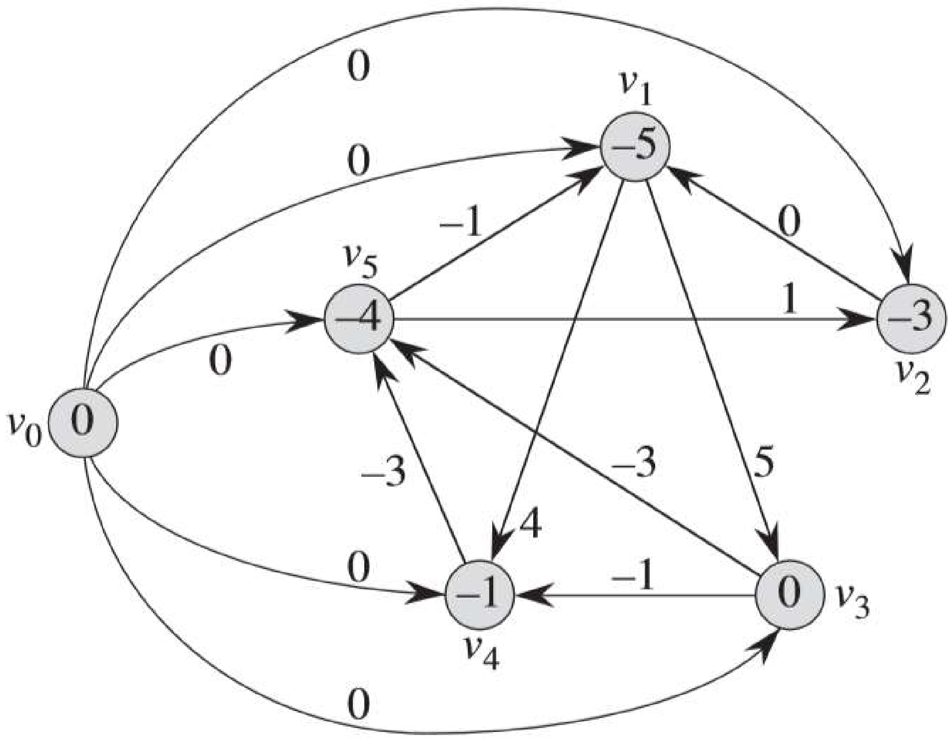
上例的差分约束系统的约束图如下：



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：



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

- 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$\omega(v_i,v_j) = b_k$， $\omega(v_0,v_j) = 0$

定理24.9 给定差分约束系统Ax≤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

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

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：

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

因此，令 $x_i = \delta(v_0, v_i)$ ，$x_j = \delta(v_0, v_j)$ 则x_i和x_j满足对应边(v_i,v_j)的差分约束条件 $x_j - x_i \le w(v_i, v_j)$。 $\omega(v_i,v_j) = b_k$

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

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

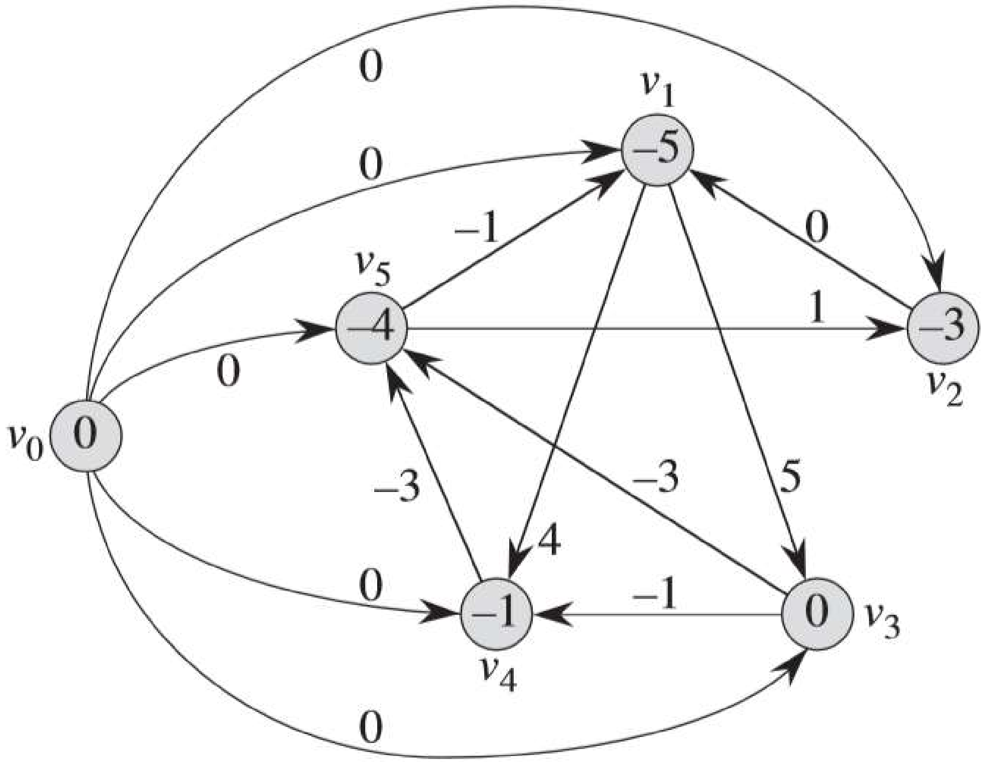
- 不等式左侧求和，等于0。($v_1=v_k$ ，所有v_i相互抵消)
- 不等式右侧求和，等于环路c的权重ω(c)，且有：0 ≤ ω(c)
- **这与c是权重为负值的环路相矛盾。**故该组不等式无解。

求解差分约束系统：

由定理24.9可得，可以使用Bellman-Ford算法来求解差分约束系统（思考为什么是Bellman-Ford算法？）。

约束图中含有从源结点v_0到其他所有结点的边,若存在权重为负值的环路，则都可以从结点v_0到达。则，

- 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。



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