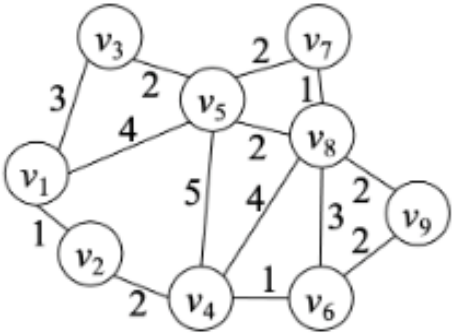
3-6 单源最短路

程龚

你能解释这些概念吗？

- 赋权图
  - 赋权函数、权
- 邻接矩阵
- 关联矩阵
- 邻接表



$$\begin{pmatrix} \infty & 1 & 3 & \infty & 4 & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & \infty & \infty & \infty & \infty & \infty \\ 3 & \infty & \infty & \infty & 2 & \infty & \infty & \infty & \infty \\ \infty & 2 & \infty & \infty & 5 & 1 & \infty & 4 & \infty \\ 4 & \infty & 2 & 5 & \infty & \infty & 2 & 2 & \infty \\ \infty & \infty & \infty & 1 & \infty & \infty & \infty & 3 & 2 \\ \infty & \infty & \infty & \infty & 2 & \infty & \infty & 1 & \infty \\ \infty & \infty & \infty & 4 & 2 & 3 & 1 & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty & 2 & \infty & 2 & \infty \end{pmatrix}$$

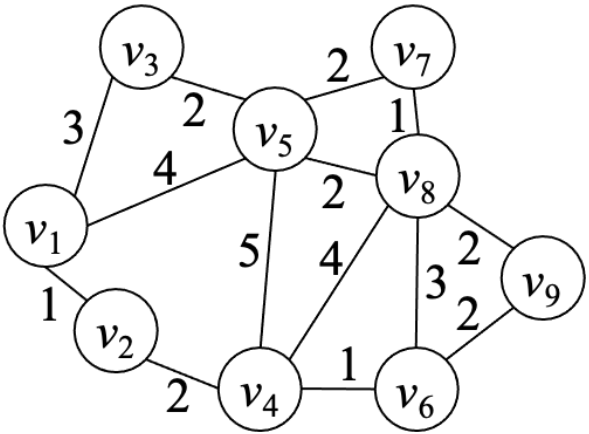
顶点	$\langle$邻点, 边权$\rangle$ 列表
v_1	$\langle v_2, 1\rangle$, $\langle v_3, 3\rangle$, $\langle v_5, 4\rangle$
v_2	$\langle v_1, 1\rangle$, $\langle v_4, 2\rangle$
v_3	$\langle v_1, 3\rangle$, $\langle v_5, 2\rangle$
v_4	$\langle v_2, 2\rangle$, $\langle v_5, 5\rangle$, $\langle v_6, 1\rangle$, $\langle v_8, 4\rangle$
v_5	$\langle v_1, 4\rangle$, $\langle v_3, 2\rangle$, $\langle v_4, 5\rangle$, $\langle v_7, 2\rangle$, $\langle v_8, 2\rangle$
v_6	$\langle v_4, 1\rangle$, $\langle v_8, 3\rangle$, $\langle v_9, 2\rangle$
v_7	$\langle v_5, 2\rangle$, $\langle v_8, 1\rangle$
v_8	$\langle v_4, 4\rangle$, $\langle v_5, 2\rangle$, $\langle v_6, 3\rangle$, $\langle v_7, 1\rangle$, $\langle v_9, 2\rangle$
v_9	$\langle v_6, 2\rangle$, $\langle v_8, 2\rangle$

你能解释这些概念吗？

- （赋权）长度

- 最短路
- 距离

- 离心率
- 中心点
- 半径
- 中心
- 边缘点
- 直径



你能解释这些概念吗？

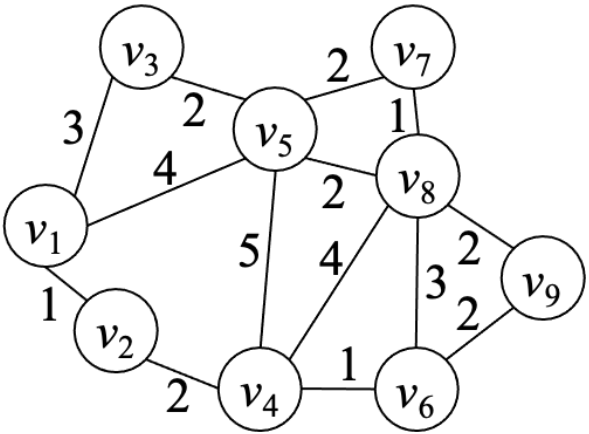
- （赋权）长度

- 最短路
- 距离

思考题 6.1 (◇) 对于连通赋权图 $G = \langle V, E, w \rangle$，距离函数 $dist$ 满足三角不等式吗：

$$\forall u, v, w \in V,\ dist(u, v) + dist(v, w) \geq dist(u, w)\,. \tag{6.2}$$

- 离心率
- 中心点
- 半径
- 中心
- 边缘点
- 直径



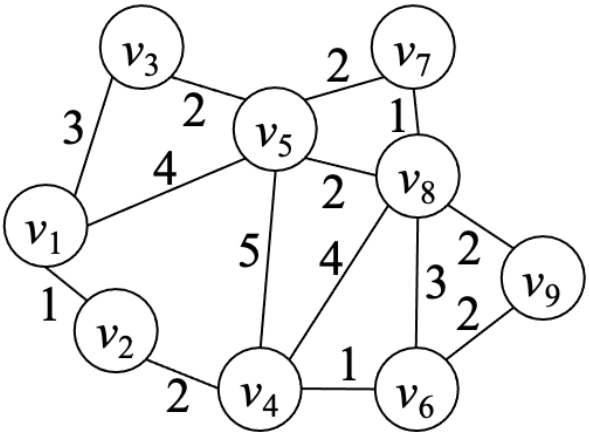
你理解这个算法了吗？

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

```
u.d ← 0;
while Q ≠ ∅ do
    v ← arg min_{w∈Q} w.d;
    Q ← Q \ {v};
    foreach (v, w) ∈ E do
        if w.d > v.d + w((v, w)) then
            w.d ← v.d + w((v, w));
```



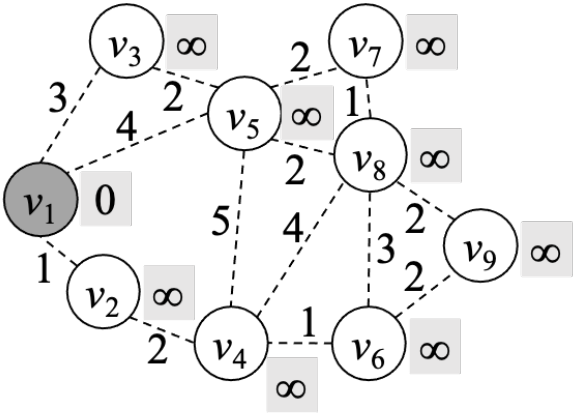
你理解这个算法了吗？

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

```
u.d ← 0;
while Q ≠ ∅ do
    v ← arg min_{w∈Q} w.d;
    Q ← Q \ {v};
    foreach (v, w) ∈ E do
        if w.d > v.d + w((v, w)) then
            w.d ← v.d + w((v, w));
```



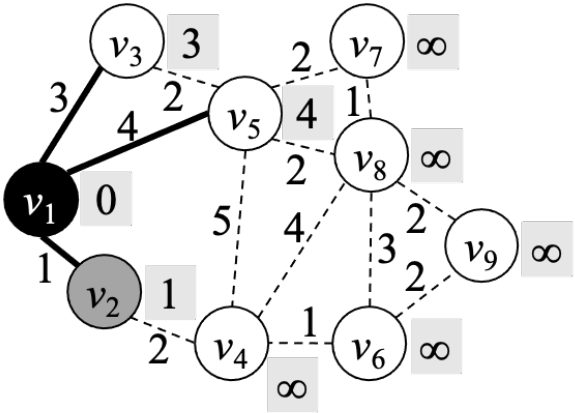
你理解这个算法了吗？

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

```
u.d ← 0;
while Q ≠ ∅ do
    v ← arg min_{w∈Q} w.d;
    Q ← Q \ {v};
    foreach (v, w) ∈ E do
        if w.d > v.d + w((v, w)) then
            w.d ← v.d + w((v, w));
```



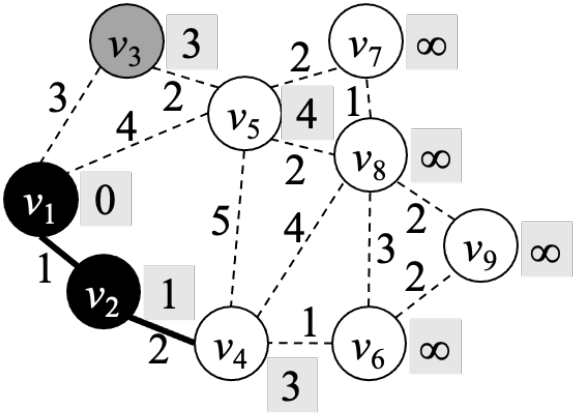
你理解这个算法了吗？

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

```
u.d ← 0;
while Q ≠ ∅ do
    v ← arg min_{w∈Q} w.d;
    Q ← Q \ {v};
    foreach (v, w) ∈ E do
        if w.d > v.d + w((v, w)) then
            w.d ← v.d + w((v, w));
```



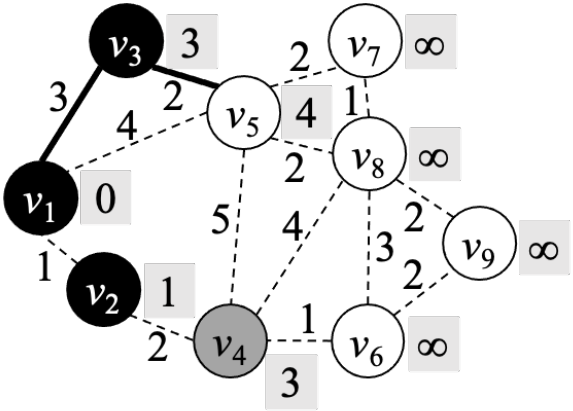
你理解这个算法了吗？

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

```
u.d ← 0;
while Q ≠ ∅ do
    v ← arg min_{w∈Q} w.d;
    Q ← Q \ {v};
    foreach (v, w) ∈ E do
        if w.d > v.d + w((v, w)) then
            w.d ← v.d + w((v, w));
```



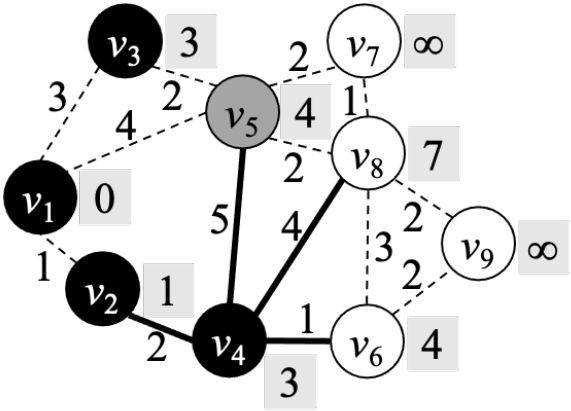
你理解这个算法了吗？

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

```
u.d ← 0;
while Q ≠ ∅ do
    v ← arg min_{w∈Q} w.d;
    Q ← Q \ {v};
    foreach (v, w) ∈ E do
        if w.d > v.d + w((v, w)) then
            w.d ← v.d + w((v, w));
```



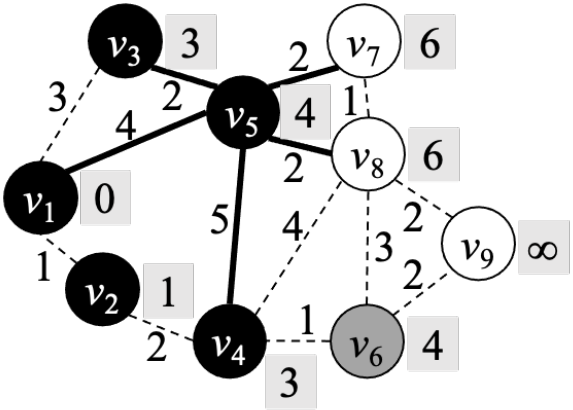
你理解这个算法了吗？

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

```
u.d ← 0;
while Q ≠ ∅ do
    v ← arg min_{w∈Q} w.d;
    Q ← Q \ {v};
    foreach (v, w) ∈ E do
        if w.d > v.d + w((v, w)) then
            w.d ← v.d + w((v, w));
```



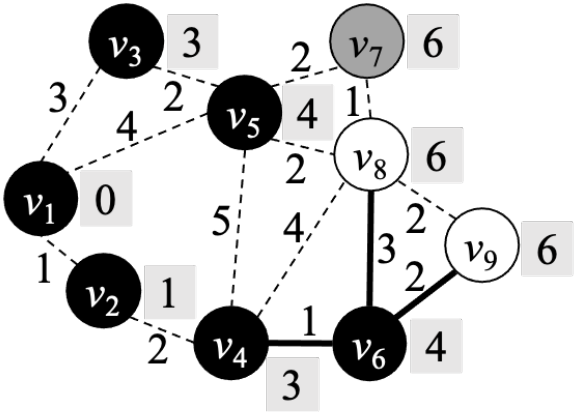
你理解这个算法了吗？

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

```
u.d ← 0;
while Q ≠ ∅ do
    v ← arg min_{w∈Q} w.d;
    Q ← Q \ {v};
    foreach (v, w) ∈ E do
        if w.d > v.d + w((v, w)) then
            w.d ← v.d + w((v, w));
```



你理解这个算法了吗？

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

$u.\mathrm{d} \leftarrow 0$;

while $Q \neq \emptyset$ **do**

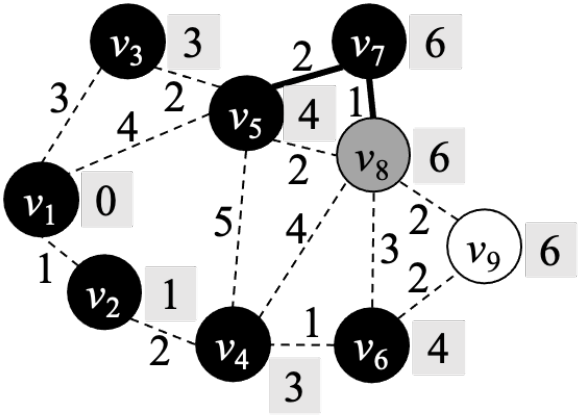
  $v \leftarrow \arg\min_{w \in Q} w.\mathrm{d}$;

  $Q \leftarrow Q \setminus \{v\}$;

  foreach $(v, w) \in E$ **do**

    if $w.\mathrm{d} > v.\mathrm{d} + w((v, w))$ **then**

      $w.\mathrm{d} \leftarrow v.\mathrm{d} + w((v, w))$;



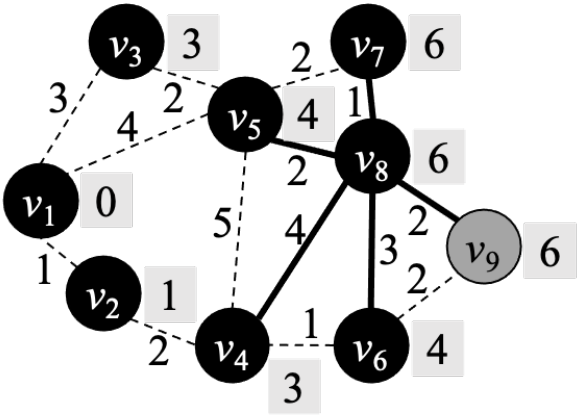
你理解这个算法了吗？

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

$u.\mathrm{d} \leftarrow 0$;
while $Q \neq \emptyset$ **do**
  $v \leftarrow \arg\min_{w \in Q} w.\mathrm{d}$;
  $Q \leftarrow Q \setminus \{v\}$;
  foreach $(v, w) \in E$ **do**
    if $w.\mathrm{d} > v.\mathrm{d} + w((v, w))$ **then**
      $w.\mathrm{d} \leftarrow v.\mathrm{d} + w((v, w))$;



你理解这个算法了吗？

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

$u.\mathrm{d} \leftarrow 0$;

while $Q \neq \emptyset$ **do**

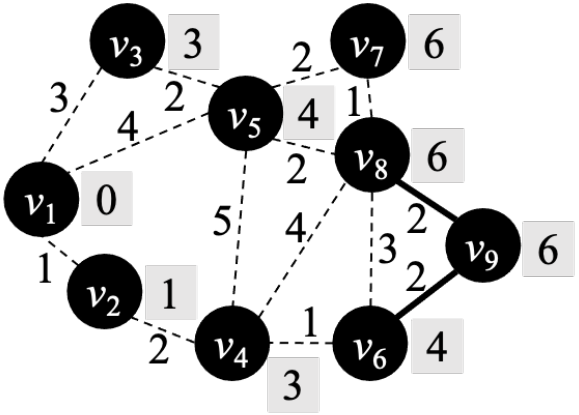
  $v \leftarrow \arg\min_{w \in Q} w.\mathrm{d}$;

  $Q \leftarrow Q \setminus \{v\}$;

  foreach $(v, w) \in E$ **do**

    if $w.\mathrm{d} > v.\mathrm{d} + w((v, w))$ **then**

      $w.\mathrm{d} \leftarrow v.\mathrm{d} + w((v, w))$;



你理解这个算法了吗？

戴克斯特拉算法只适用于边权非负的赋权图。

思考题 6.2 对于含负权边的赋权图，戴克斯特拉算法运行结束时，和顶点 u 连通的顶点的 d 属性值未必为其和 u 间的距离，你能举例说明吗？

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

```
u.d ← 0;
while Q ≠ ∅ do
    v ← arg min_{w∈Q} w.d;
    Q ← Q \ {v};
    foreach (v, w) ∈ E do
        if w.d > v.d + w((v, w)) then
            w.d ← v.d + w((v, w));
```

请证明下述定理

定理 6.1 戴克斯特拉算法每轮 while 循环条件判定前，已从集合 Q 中删除的每个顶点 v 满足 $v.\mathrm{d} = dist(u, v)$。

- 反证法
  - v的含义？
  - y的含义？
  - x的含义？

$$
\begin{aligned}
y.\mathrm{d} &\le x.\mathrm{d} + w((x,y)) \\
&= dist(u,y) \\
&\le \underline{dist(u,v)} \\
&\le \underline{v.\mathrm{d}} \\
&\le y.\mathrm{d}\,.
\end{aligned}
$$

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w\rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

```
u.d ← 0;
while Q ≠ ∅ do
    v ← arg min_{w∈Q} w.d;
    Q ← Q \ {v};
    foreach (v, w) ∈ E do
        if w.d > v.d + w((v, w)) then
            w.d ← v.d + w((v, w));
```

你理解这个算法了吗？

思考题 6.4 负权边的存在对上述证明过程有何影响？

- 反证法
  - v的含义？
  - y的含义？
  - x的含义？

$$
\begin{aligned}
y.\mathrm{d} &\le x.\mathrm{d} + w((x,y)) \\
&= dist(u,y) \\
&\le dist(u,v) \\
&\le v.\mathrm{d} \\
&\le y.\mathrm{d}\,.
\end{aligned}
$$

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w\rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

```
u.d ← 0;
while Q ≠ ∅ do
    v ← arg min_{w∈Q} w.d;
    Q ← Q \ {v};
    foreach (v, w) ∈ E do
        if w.d > v.d + w((v, w)) then
            w.d ← v.d + w((v, w));
```

你理解这个算法了吗？

思考题 6.5 为什么戴克斯特拉算法从集合 Q 中删除顶点的顺序是按到顶点 u 的距离从小到大？

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

```
u.d ← 0;
while Q ≠ ∅ do
    v ← arg min_{w∈Q} w.d;
    Q ← Q \ {v};
    foreach (v, w) ∈ E do
        if w.d > v.d + w((v, w)) then
            w.d ← v.d + w((v, w));
```

你能解释这个算法的时间复杂度吗？

- 对于阶为n、边数为m的赋权图，戴克斯特拉算法的时间复杂度为$O((n+m)\log n)$

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

```
u.d ← 0;
while Q ≠ ∅ do
    v ← arg min_{w∈Q} w.d;
    Q ← Q \ {v};
    foreach (v, w) ∈ E do
        if w.d > v.d + w((v, w)) then
            w.d ← v.d + w((v, w));
```

你理解Relax了吗？

算法 6.1: 戴克斯特拉算法

输入: 赋权图 $G = \langle V, E, w \rangle$，顶点 u

初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

```
u.d ← 0;
while Q ≠ ∅ do
    v ← arg min_{w∈Q} w.d;
    Q ← Q \ {v};
    foreach (v, w) ∈ E do
        if w.d > v.d + w((v, w)) then
            w.d ← v.d + w((v, w));
```

```
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)
```

你理解这些性质了吗？

Triangle inequality (Lemma 24.10)
For any edge $(u, v) \in E$, we have $\delta(s, v) \le \delta(s, u) + w(u, v)$.

Upper-bound property (Lemma 24.11)
We always have $v.d \ge \delta(s, v)$ for all vertices $v \in V$, and once $v.d$ achieves the value $\delta(s, v)$, it never changes.

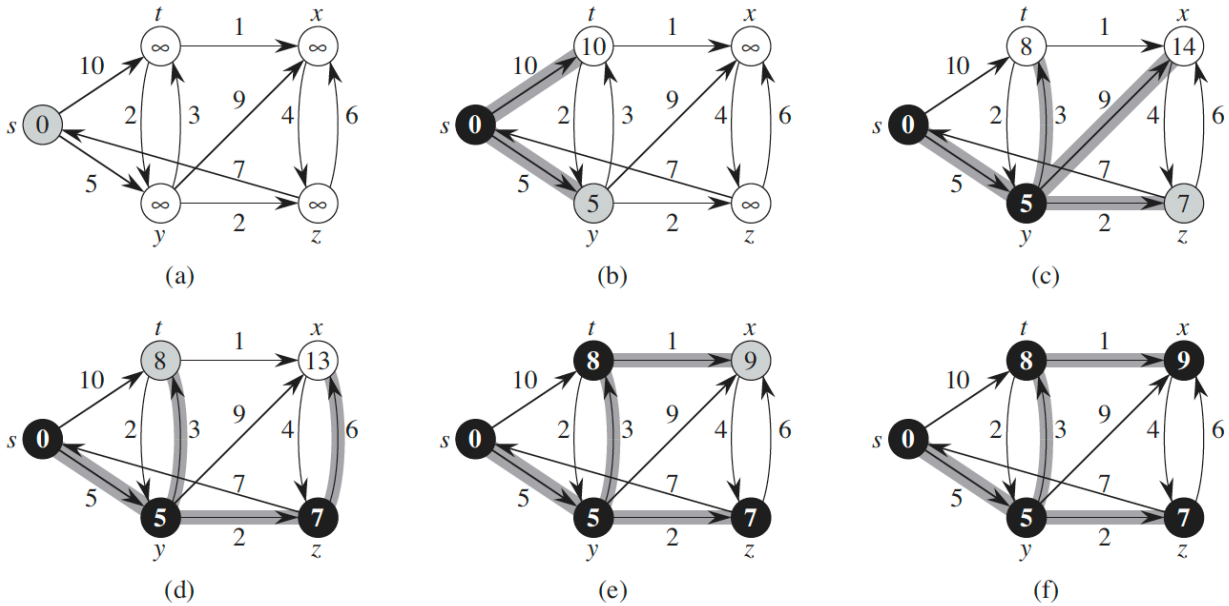
No-path property (Corollary 24.12)
If there is no path from s to v, then we always have $v.d = \delta(s, v) = \infty$.

Convergence property (Lemma 24.14)
If $s \leadsto u \to v$ is a shortest path in G for some $u, v \in V$, and if $u.d = \delta(s, u)$ at any time prior to relaxing edge (u, v), then $v.d = \delta(s, v)$ at all times afterward.

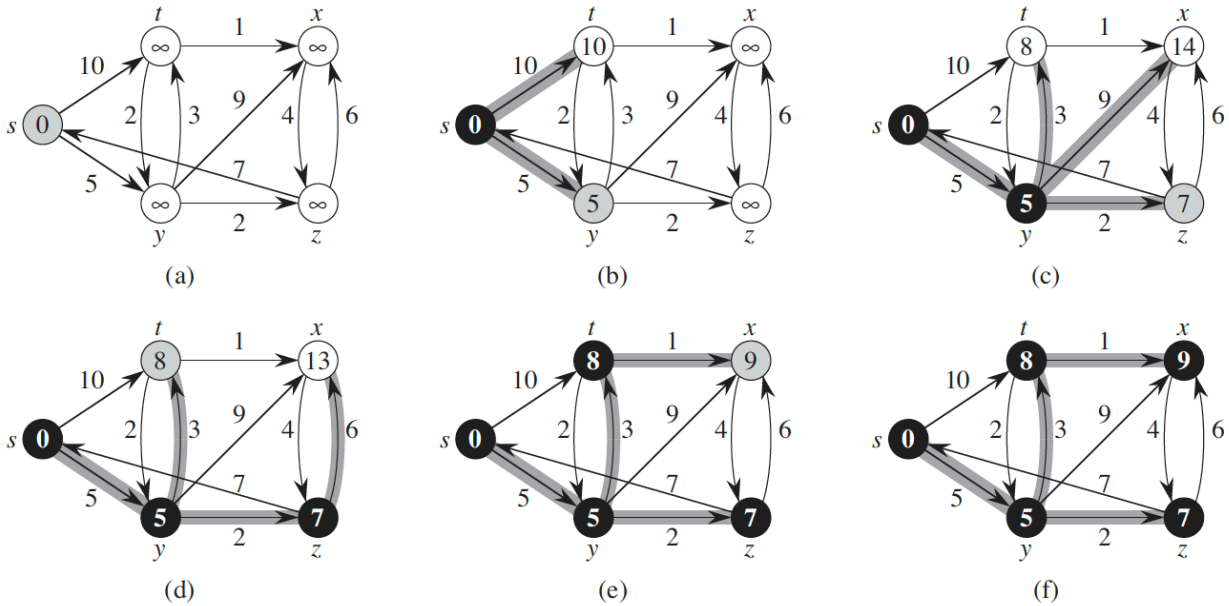
Path-relaxation property (Lemma 24.15)
If $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = \delta(s, v_k)$. This property holds regardless of any other relaxation steps that occur, even if they are intermixed with relaxations of the edges of p.

Predecessor-subgraph property (Lemma 24.17)
Once $v.d = \delta(s, v)$ for all $v \in V$, the predecessor subgraph is a shortest-paths tree rooted at s.

你理解这些性质了吗？



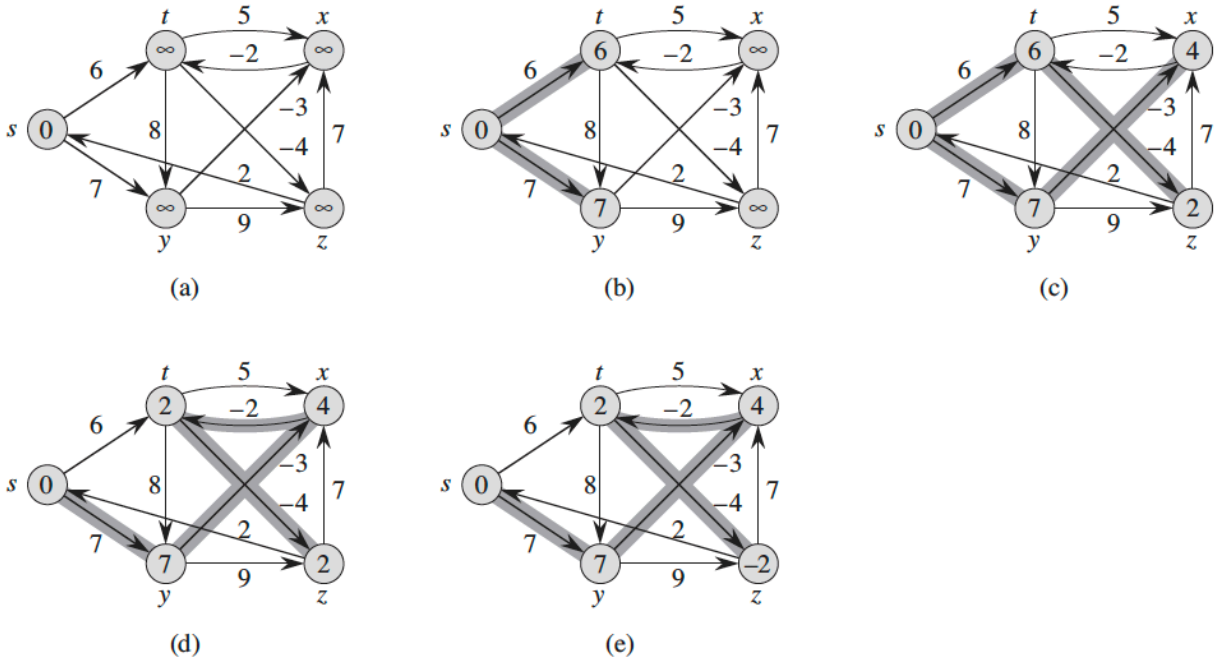
至少需要多少次Relax?



你理解这个算法了吗？

```
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
```

你理解这个算法了吗？



你理解这个算法了吗？

- 你能比较这两个算法Relax的顺序和次数吗？

```
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
```

```
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)
```

你理解这个算法了吗？

- 你能比较这两个算法的循环不变式吗？
  （哪些顶点满足 $v.d = \delta(s, v)$？）

Path-relaxation property (Lemma 24.15)
If $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = \delta(s, v_k)$. This property holds regardless of any other relaxation steps that occur, even if they are intermixed with relaxations of the edges of p.

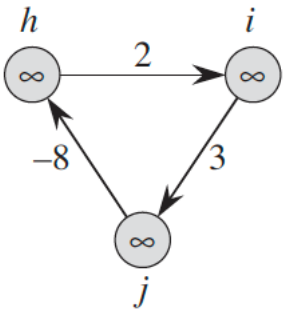
```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

```
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)
```

你理解这个算法了吗？

- 什么是负权圈（negative-weight cycle）？
- 这个算法一定能发现图中的负权圈吗？

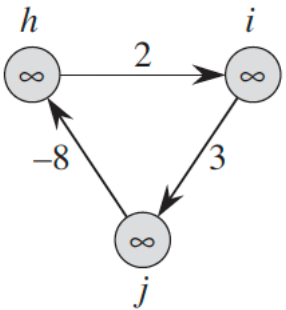
```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```



你理解这个算法了吗？

- 什么是负权圈（negative-weight cycle）？
- 这个算法一定能发现图中的负权圈吗？
- 为什么一定能发现从s可达的负权圈？

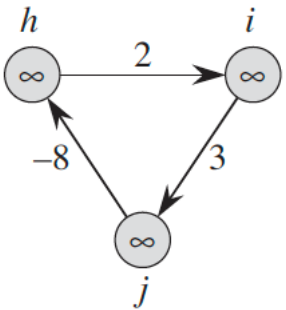
```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```



你理解这个算法了吗？

- 什么是负权圈（negative-weight cycle）？
- 这个算法一定能发现图中的负权圈吗？
- 为什么一定能发现从s可达的负权圈？
- 你能改造这个算法，让它也能发现从s不可达的负权圈吗？

```
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
```



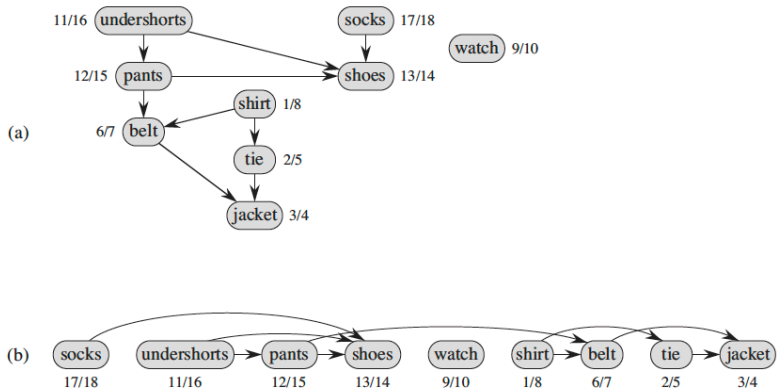
你能解释这个算法的时间复杂度吗？

- 对于阶为n、边数为m的赋权图，贝尔曼-福特算法的时间复杂度为$O(nm)$

```
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
```

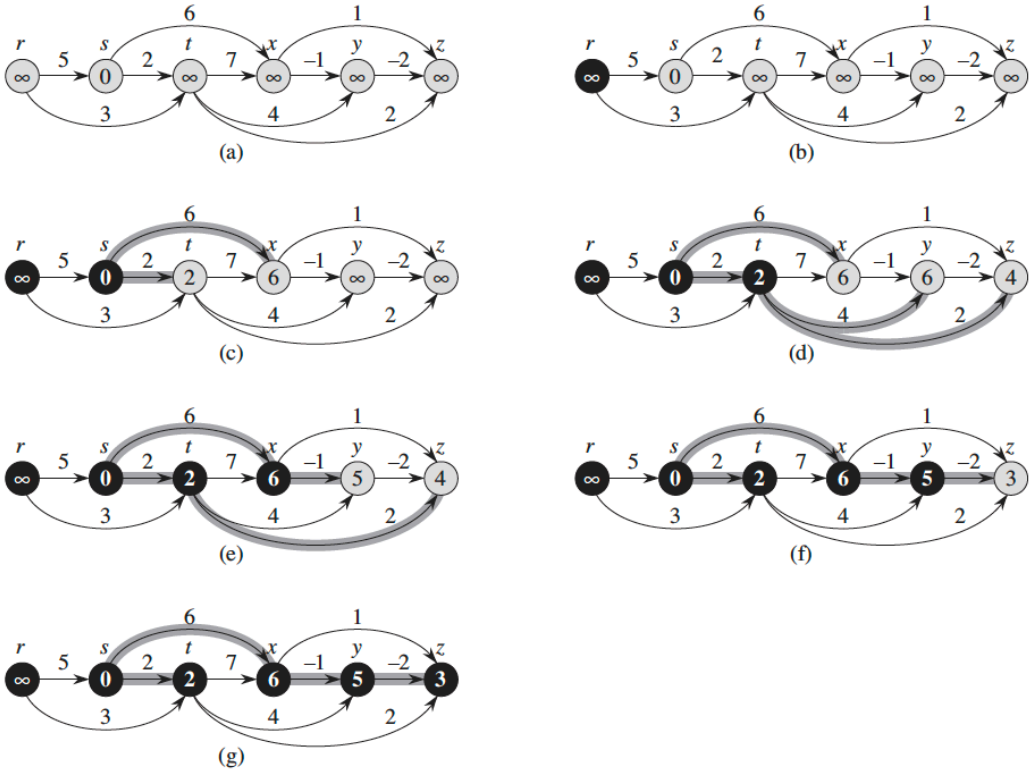
你理解这个算法了吗？

- 什么是DAG？什么是拓扑序？




```
DAG-SHORTEST-PATHS(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

你理解这个算法了吗？



你理解这个算法了吗？

- 你能比较这两个算法Relax的顺序和次数吗？

```
DAG-SHORTEST-PATHS(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

```
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
```

你理解这个算法了吗？

- 你能比较这两个算法的循环不变式吗？
  （哪些顶点满足 $v.d = \delta(s, v)$？）

Path-relaxation property (Lemma 24.15)
If $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = \delta(s, v_k)$. This property holds regardless of any other relaxation steps that occur, even if they are intermixed with relaxations of the edges of p.

```
DAG-SHORTEST-PATHS(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

```
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
```

你能解释这个算法的时间复杂度吗？

- 对于阶为n、边数为m的赋权图，Dag-Shortest-Paths算法的时间复杂度为$O(n+m)$

```
DAG-SHORTEST-PATHS(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

我们为什么不讨论single-pair shortest-path problem？

单源最短路的应用

- 如何更新设备最划算？

	2021	2022	2023	2024	2025
买入价	11	12	13	14	14

	1年后	2年后	3年后	4年后	5年后
卖出价	4	3	2	1	0

	第1年	第2年	第3年	第4年	第5年
保养费	5	6	8	11	18

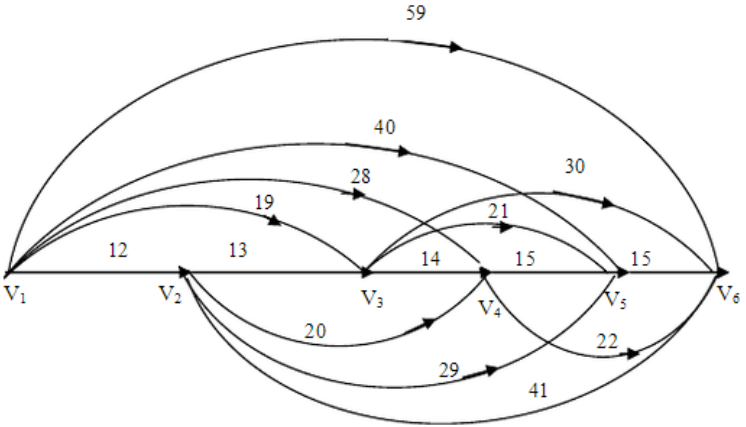
单源最短路的应用

- 如何更新设备最划算？

	2021	2022	2023	2024	2025
买入价	11	12	13	14	14

	1年后	2年后	3年后	4年后	5年后
卖出价	4	3	2	1	0

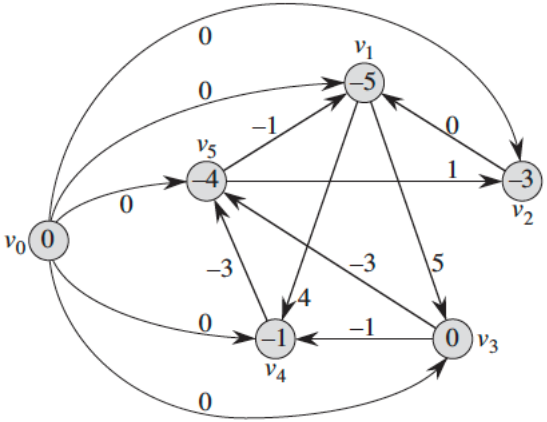
	第1年	第2年	第3年	第4年	第5年
保养费	5	6	8	11	18



单源最短路的应用

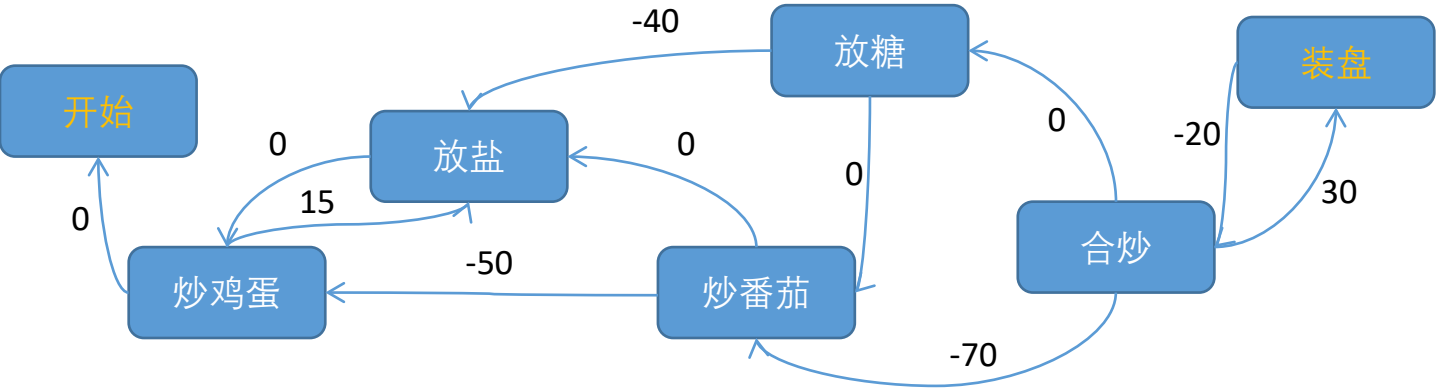
- 你能解释这些术语吗？
  - system of difference constraints
  - constraint graph

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

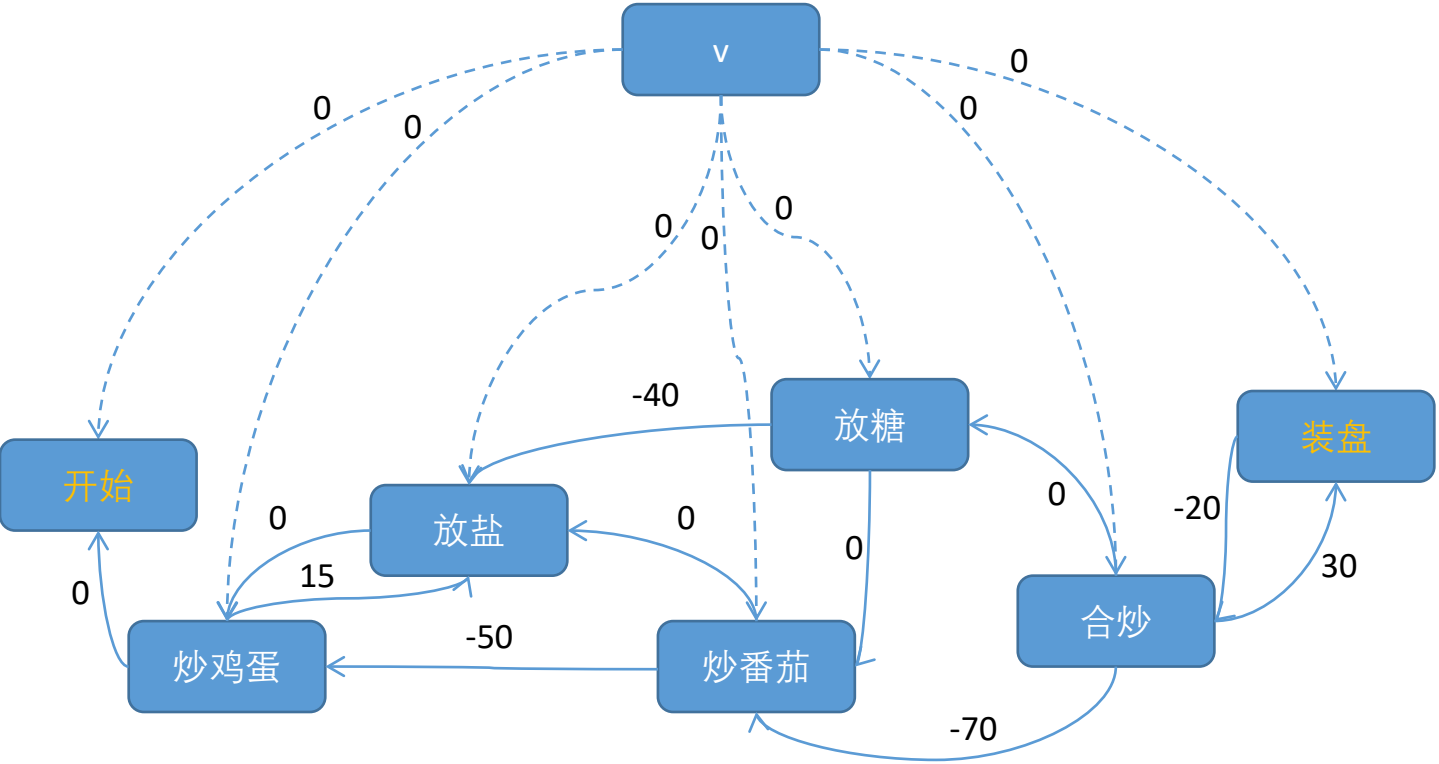


单源最短路的应用

- 你会做番茄炒蛋吗？
  - 顺序：先把鸡蛋炒熟，再炒番茄，再合起来炒
  - 要点：
    - 炒鸡蛋至少要50秒才能熟
    - 鸡蛋下锅的15秒内要放盐才能入味
    - 炒番茄至少要70秒才能出汁
    - 炒番茄时要放糖，但和之前放盐的时间要间隔至少40秒
    - 合起来炒至少要20秒才能混味，但不能超过30秒，否则就稀烂了
    - ……
- 如果一个新手要做番茄炒蛋，你能为他列出一份具体的时间表吗？（第几秒时做什么）



单源最短路的应用



OT

- 单源最短路问题有很多并行算法，例如Δ-stepping算法、Radius Stepping算法等，请调研至少2种算法（其中至多1种来自上述例子），结合例子介绍算法的设计与分析，比较异同并分析优劣。