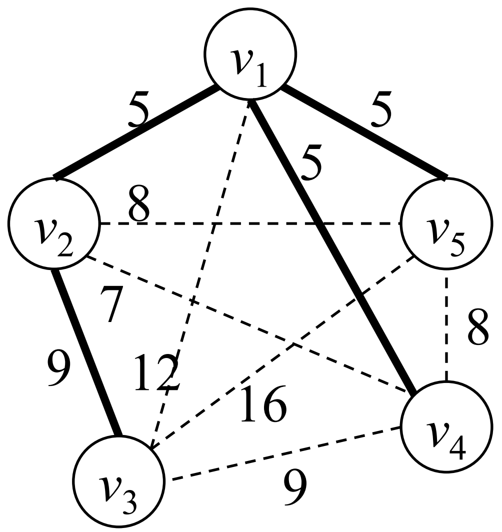
第6章 赋权图

程龚
南京大学 计算机学院
gcheng@nju.edu.cn
http://ws.nju.edu.cn/~gcheng

本章内容

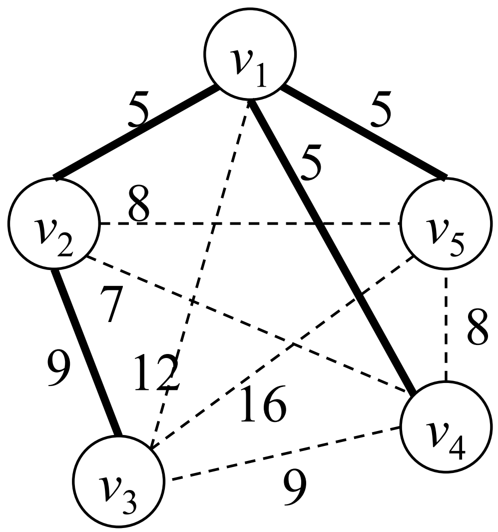
- 第6.1节 赋权图和距离
- 第6.2节 最小生成树
  - 第6.2.1节 理论
  - **第6.2.2节 算法**
- 第6.3节 赋权欧拉图
- 第6.4节 赋权哈密尔顿图

如何找出最小生成树?



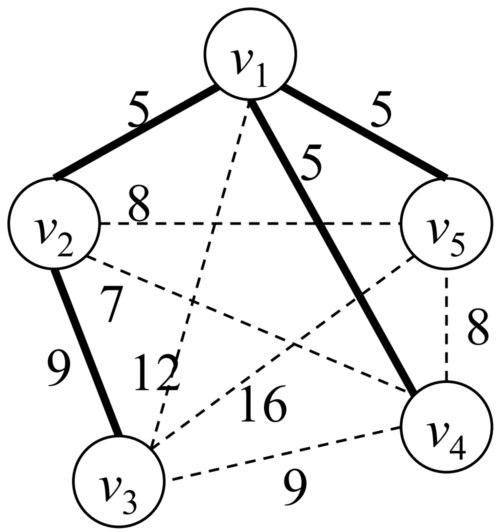
如何找出最小生成树？

1. 普里姆算法
2. 克拉斯克尔算法



如何找出最小生成树?

1. **普里姆算法**
2. 克拉斯克尔算法



普里姆算法

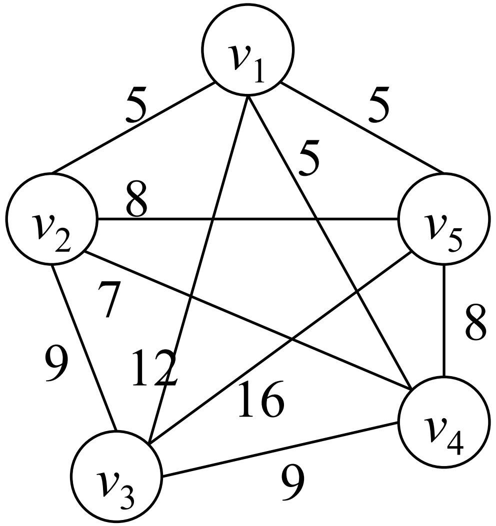
- 基本思路：逐步构造最小生成树，每步向当前树增加一条边，并尽可能选择权小的边。

算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；
集合 Q 初值为 V

$r \leftarrow V$ 中任意一个顶点;
r.mw $\leftarrow 0$;
while $Q \neq \emptyset$ **do**
  $v \leftarrow \arg\min_{u \in Q} u.\mathrm{mw}$;
  $Q \leftarrow Q \setminus \{v\}$;
  if $v \neq r$ **then**
    输出 $((v.parent, v))$;
  foreach $(v, w) \in E$ **do**
    if $w \in Q$ 且 $w((v, w)) < w.\mathrm{mw}$ **then**
      $w.\mathrm{mw} \leftarrow w((v, w))$;
      $w.\mathrm{parent} \leftarrow v$;



普里姆算法

- 每个顶点的
  - mw属性：实数型变量，表示该顶点与当前树中顶点间的边的最小权，初值为∞
  - parent属性：初值为null的顶点型变量，表示上述这条权最小的边在当前树中的端点

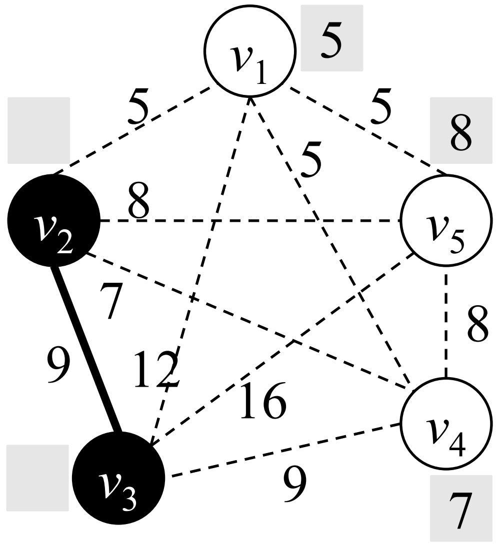
算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；集合 Q 初值为 V

$r \leftarrow V$ 中任意一个顶点;
$r.\text{mw} \leftarrow 0$;
while $Q \neq \emptyset$ **do**
    $v \leftarrow \arg\min_{u \in Q} u.\text{mw}$;
    $Q \leftarrow Q \setminus \{v\}$;
    if $v \neq r$ **then**
        输出 $((v.parent, v))$;
    foreach $(v, w) \in E$ **do**
        if $w \in Q$ 且 $w((v, w)) < w.\text{mw}$ **then**
            $w.\text{mw} \leftarrow w((v, w))$;
            $w.\text{parent} \leftarrow v$;

黑色顶点：已增加到当前树中的顶点
粗实线：已增加到当前树中的边



普里姆算法

- 从仅含V中任意一个顶点r的平凡树开始。

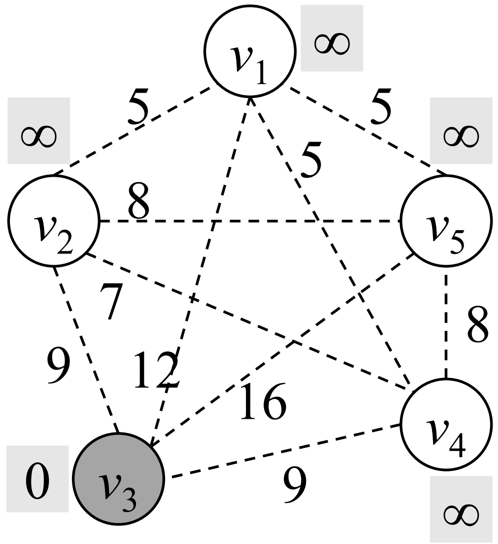
算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；集合 Q 初值为 V

$r \leftarrow V$ 中任意一个顶点;
r.mw $\leftarrow 0$;
while $Q \neq \emptyset$ **do**
    $v \leftarrow \arg\min_{u \in Q} u.\mathrm{mw}$;
    $Q \leftarrow Q \setminus \{v\}$;
    if $v \neq r$ **then**
        输出 $((v.parent, v))$;
    foreach $(v, w) \in E$ **do**
        if $w \in Q$ 且 $w((v, w)) < w.\mathrm{mw}$ **then**
            $w.\mathrm{mw} \leftarrow w((v, w))$;
            $w.\mathrm{parent} \leftarrow v$;

黑色顶点：已增加到当前树中的顶点
粗实线：已增加到当前树中的边



普里姆算法

- 每轮while循环从初值为V的集合Q中选择一个顶点v删除，直至Q为空：

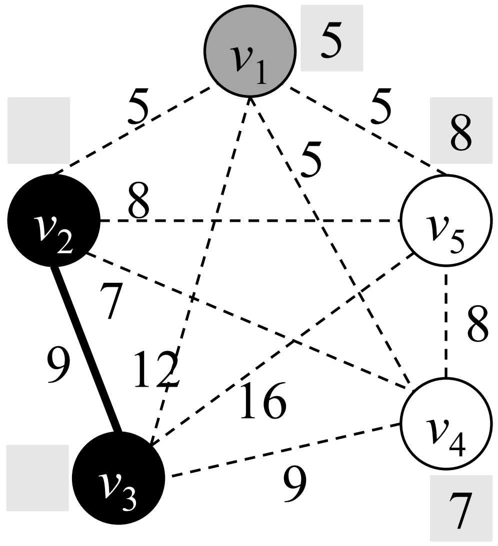
算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；集合 Q 初值为 V

$r \leftarrow V$ 中任意一个顶点;
r.mw $\leftarrow 0$;
while $Q \neq \emptyset$ **do**
  $v \leftarrow \arg\min_{u \in Q} u$.mw;
  $Q \leftarrow Q \setminus \{v\}$;
  if $v \neq r$ **then**
    输出 $((v.parent, v))$;
  foreach $(v, w) \in E$ **do**
    if $w \in Q$ 且 $w((v, w)) < w$.mw **then**
      w.mw $\leftarrow w((v, w))$;
      w.parent $\leftarrow v$;

黑色顶点：已增加到当前树中的顶点
粗实线：已增加到当前树中的边
灰色顶点：下轮while循环选择的顶点v



普里姆算法

- 每轮while循环从初值为V的集合Q中选择一个顶点v删除，直至Q为空：
  - v是Q中mw属性值最小的顶点，将边(v.parent, v)增加到当前树中

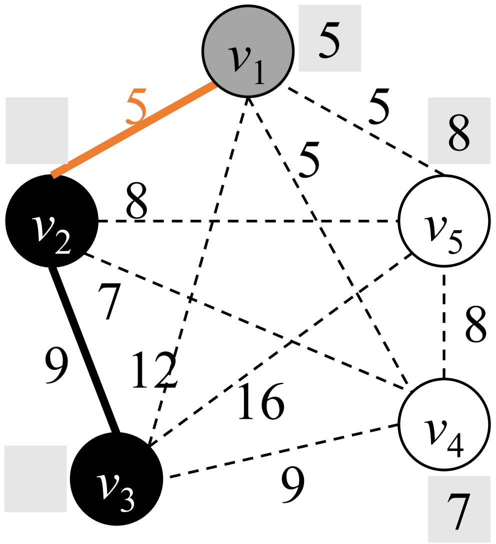
算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w\rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；集合 Q 初值为 V

```
r ← V 中任意一个顶点;
r.mw ← 0;
while Q ≠ ∅ do
    v ← arg min_{u∈Q} u.mw;
    Q ← Q \ {v};
    if v ≠ r then
        输出 ((v.parent, v));
    foreach (v, w) ∈ E do
        if w ∈ Q 且 w((v, w)) < w.mw then
            w.mw ← w((v, w));
            w.parent ← v;
```

黑色顶点：已增加到当前树中的顶点
粗实线：已增加到当前树中的边
灰色顶点：下轮while循环选择的顶点v



普里姆算法

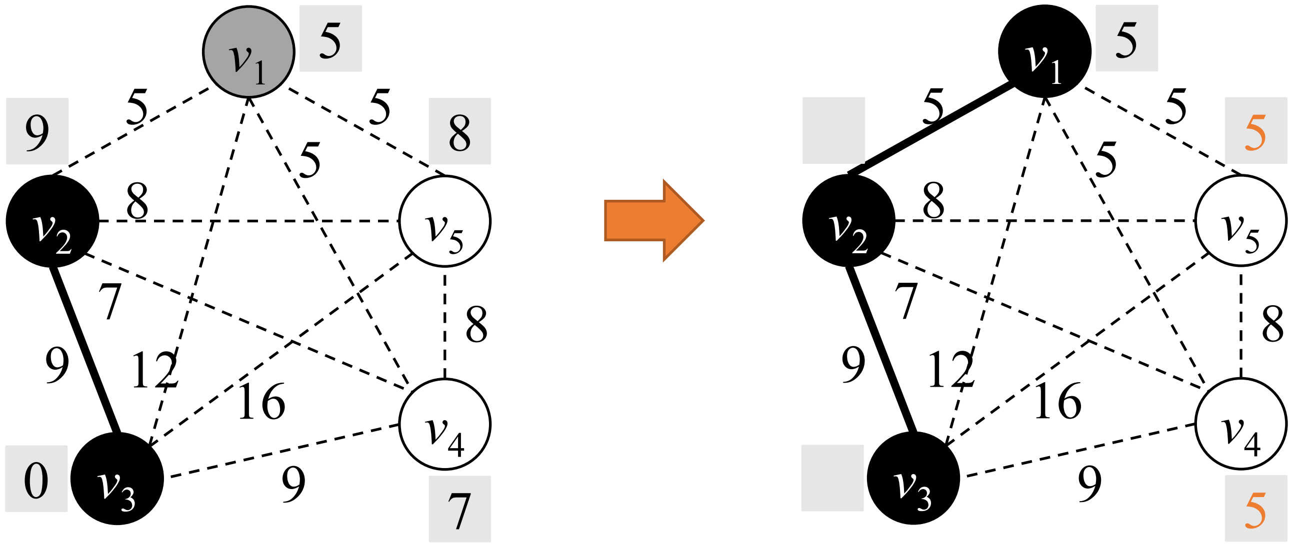
- 每轮while循环从初值为V的集合Q中选择一个顶点v删除，直至Q为空：
  - v是Q中mw属性值最小的顶点，将边(v.parent, v)增加到当前树中
  - 对于v的每个仍在Q中的邻点w：
    - 若边(v, w)的权比w与之前树中顶点间的边的最小权更小，则更新w的mw属性值和parent属性值

算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；集合 Q 初值为 V

```
r ← V 中任意一个顶点;
r.mw ← 0;
while Q ≠ ∅ do
    v ← arg min_{u∈Q} u.mw;
    Q ← Q \ {v};
    if v ≠ r then
        输出 ((v.parent, v));
    foreach (v, w) ∈ E do
        if w ∈ Q 且 w((v, w)) < w.mw then
            w.mw ← w((v, w));
            w.parent ← v;
```



普里姆算法

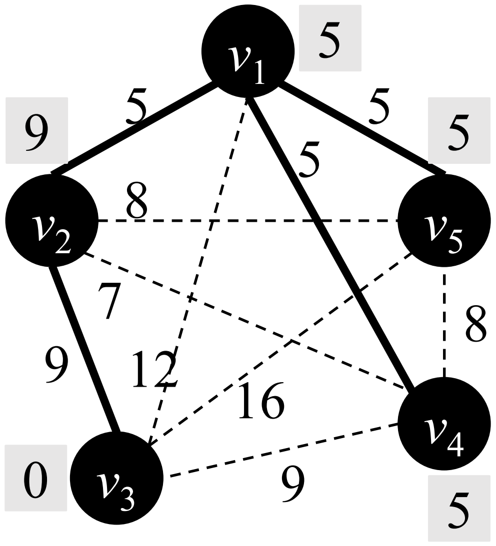
- 算法运行结束时，输出的边组成一棵最小生成树。

算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；集合 Q 初值为 V

$r \leftarrow V$ 中任意一个顶点;
$r.\mathrm{mw} \leftarrow 0$;
while $Q \neq \emptyset$ **do**
    $v \leftarrow \arg\min_{u \in Q} u.\mathrm{mw}$;
    $Q \leftarrow Q \setminus \{v\}$;
    if $v \neq r$ **then**
        输出 $((v.parent,\ v))$;
    foreach $(v, w) \in E$ **do**
        if $w \in Q$ 且 $w((v, w)) < w.\mathrm{mw}$ **then**
            $w.\mathrm{mw} \leftarrow w((v, w))$;
            $w.\mathrm{parent} \leftarrow v$;



普里姆算法

- 例如：从仅含v_3的平凡树开始，下轮while循环选择v_3

算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；集合 Q 初值为 V

$r \leftarrow V$ 中任意一个顶点;

r.mw $\leftarrow 0$;

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

  $v \leftarrow \arg\min_{u \in Q} u$.mw;

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

  if $v \neq r$ **then**

    输出 $((v.parent,\ v))$;

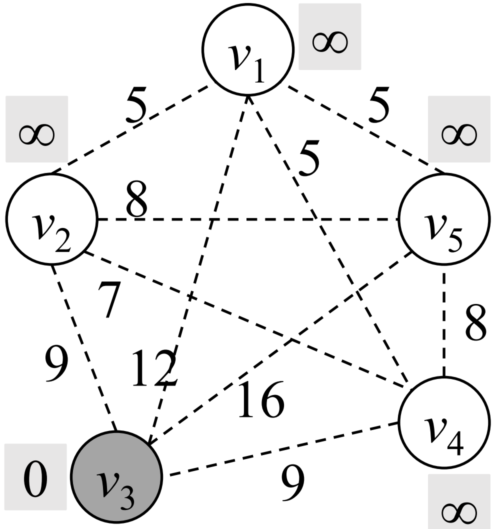
  foreach $(v, w) \in E$ **do**

    if $w \in Q$ 且 $w((v, w)) < w$.mw **then**

      w.mw $\leftarrow w((v, w))$;

      w.parent $\leftarrow v$;

黑色顶点：已增加到当前树中的顶点
粗实线：已增加到当前树中的边
灰色顶点：下轮while循环选择的顶点v



普里姆算法

- 从Q中删除v_3
- 更新v_1、v_2、v_4、v_5的mw属性值（和parent属性值）
- 下轮while循环选择v_2

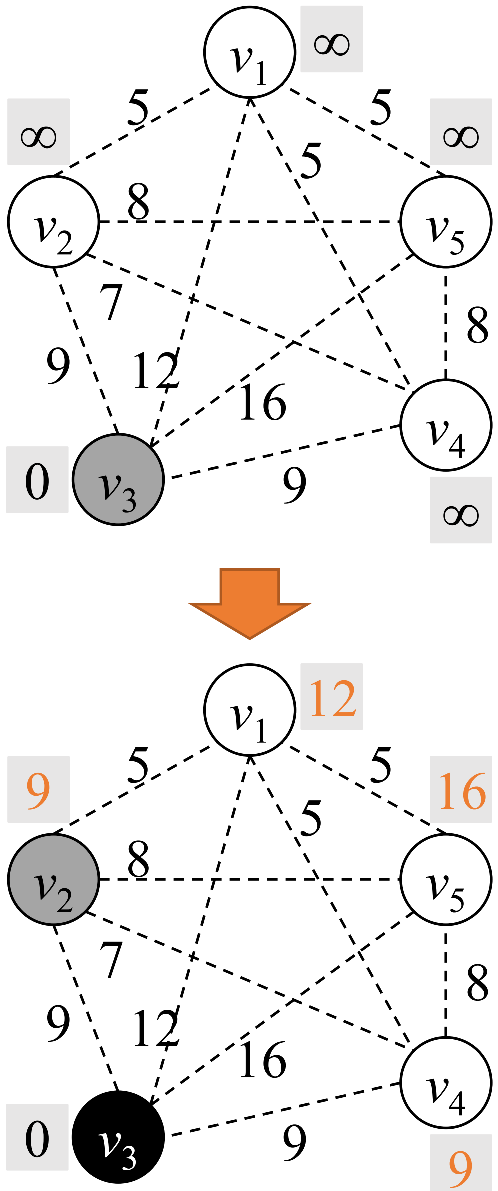
算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；集合 Q 初值为 V

$r \leftarrow V$ 中任意一个顶点;
r.mw $\leftarrow 0$;
while $Q \neq \emptyset$ **do**
  $v \leftarrow \arg\min_{u \in Q} u$.mw;
  $Q \leftarrow Q \setminus \{v\}$;
  if $v \neq r$ **then**
    输出 $((v.parent, v))$;
  foreach $(v, w) \in E$ **do**
    if $w \in Q$ 且 $w((v, w)) < w$.mw **then**
      w.mw $\leftarrow w((v, w))$;
      w.parent $\leftarrow v$;

黑色顶点：已增加到当前树中的顶点
粗实线：已增加到当前树中的边
灰色顶点：下轮while循环选择的顶点v



普里姆算法

- 从Q中删除v_2
- 将(v_3, v_2)增加到当前树中
- 更新v_1、v_4、v_5的mw属性值（和parent属性值）
- 下轮while循环选择v_1

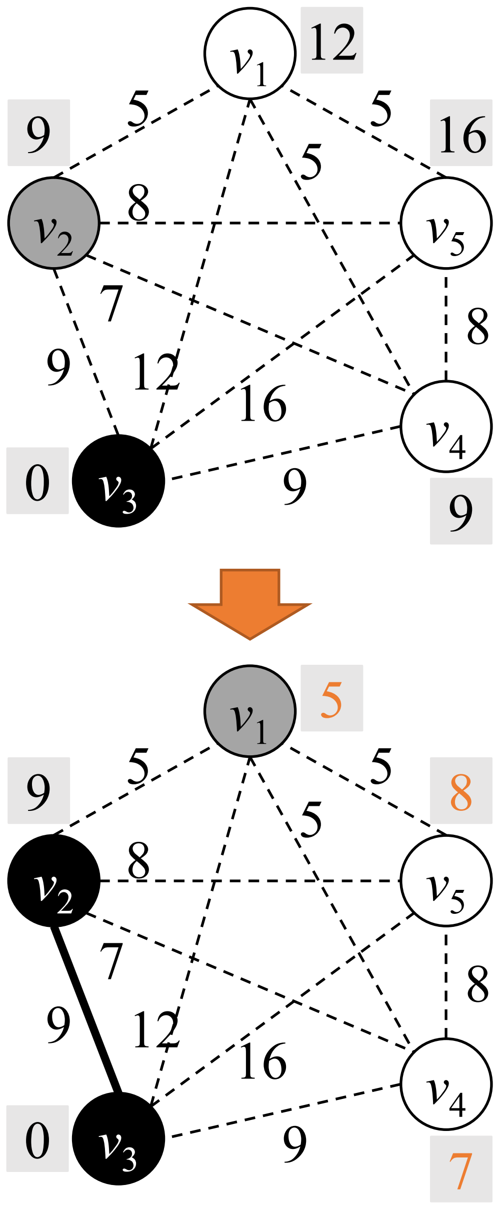
算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w\rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；集合 Q 初值为 V

$r \leftarrow V$ 中任意一个顶点;
r.mw $\leftarrow 0$;
while $Q \neq \emptyset$ **do**
  $v \leftarrow \arg\min_{u\in Q} u.\mathrm{mw}$;
  $Q \leftarrow Q \setminus \{v\}$;
  if $v \neq r$ **then**
    输出 $((v.parent, v))$;
  foreach $(v, w) \in E$ **do**
    if $w \in Q$ 且 $w((v,w)) < w.\mathrm{mw}$ **then**
      $w.\mathrm{mw} \leftarrow w((v,w))$;
      $w.\mathrm{parent} \leftarrow v$;

黑色顶点：已增加到当前树中的顶点
粗实线：已增加到当前树中的边
灰色顶点：下轮while循环选择的顶点v



普里姆算法

- 从Q中删除v_1
- 将(v_2, v_1)增加到当前树中
- 更新v_4、v_5的mw属性值（和parent属性值）
- 下轮while循环选择v_4

算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；集合 Q 初值为 V

$r \leftarrow V$ 中任意一个顶点;

$r.\text{mw} \leftarrow 0$;

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

  $v \leftarrow \arg\min_{u \in Q} u.\text{mw}$;

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

  if $v \neq r$ **then**

    输出 $((v.parent, v))$;

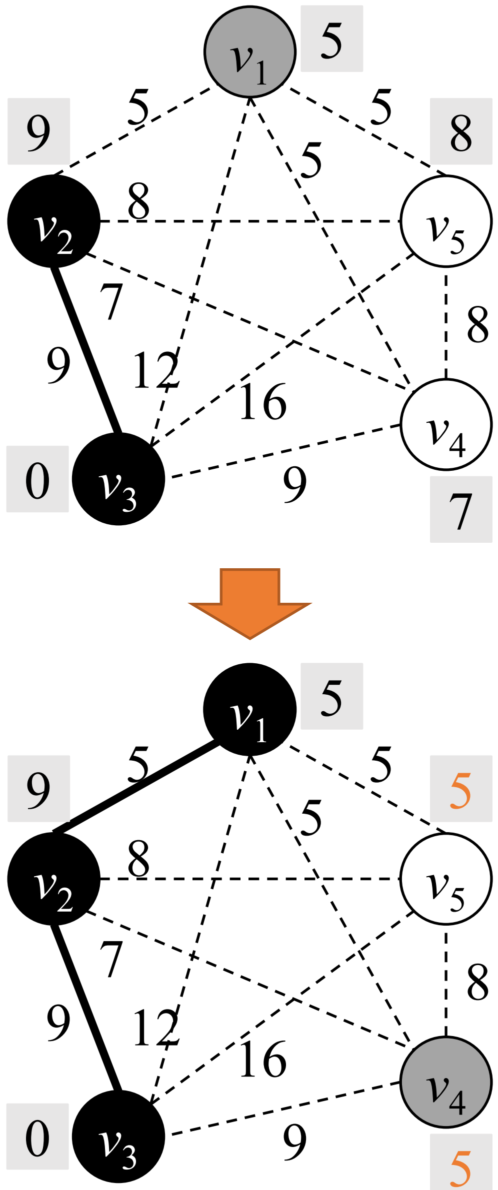
  foreach $(v, w) \in E$ **do**

    if $w \in Q$ 且 $w((v, w)) < w.\text{mw}$ **then**

      $w.\text{mw} \leftarrow w((v, w))$;

      $w.\text{parent} \leftarrow v$;

黑色顶点：已增加到当前树中的顶点
粗实线：已增加到当前树中的边
灰色顶点：下轮while循环选择的顶点v



普里姆算法

- 从Q中删除v_4
- 将(v_1, v_4)增加到当前树中
- 下轮while循环选择v_5

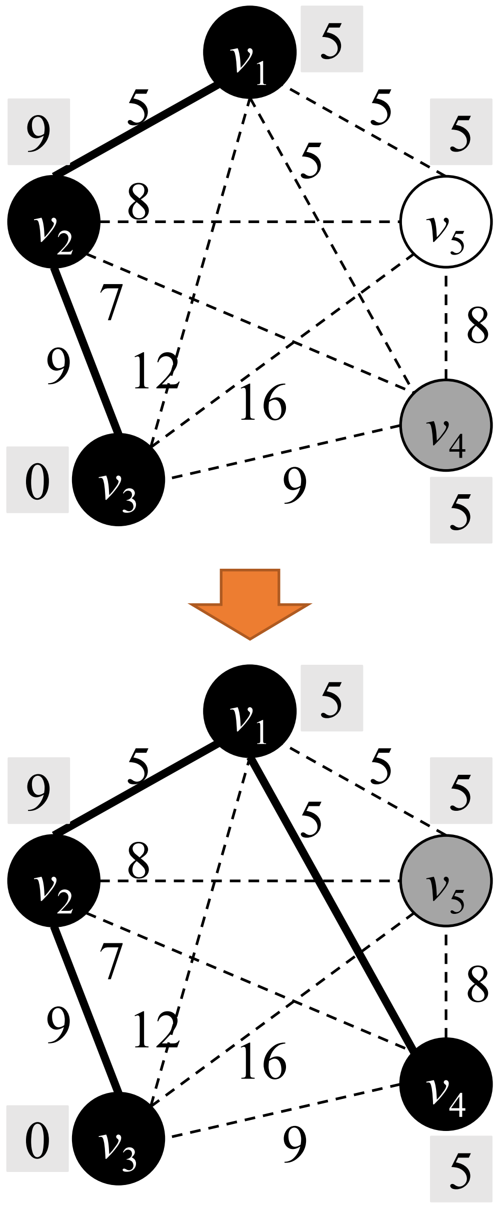
算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；集合 Q 初值为 V

```
r ← V 中任意一个顶点;
r.mw ← 0;
while Q ≠ ∅ do
    v ← arg min_{u∈Q} u.mw;
    Q ← Q \ {v};
    if v ≠ r then
        输出 ((v.parent, v));
    foreach (v, w) ∈ E do
        if w ∈ Q 且 w((v, w)) < w.mw then
            w.mw ← w((v, w));
            w.parent ← v;
```

黑色顶点：已增加到当前树中的顶点
粗实线：已增加到当前树中的边
灰色顶点：下轮while循环选择的顶点v



普里姆算法

- 从Q中删除v_5
- 将(v_1, v_5)增加到当前树中
- 下轮while循环Q为空

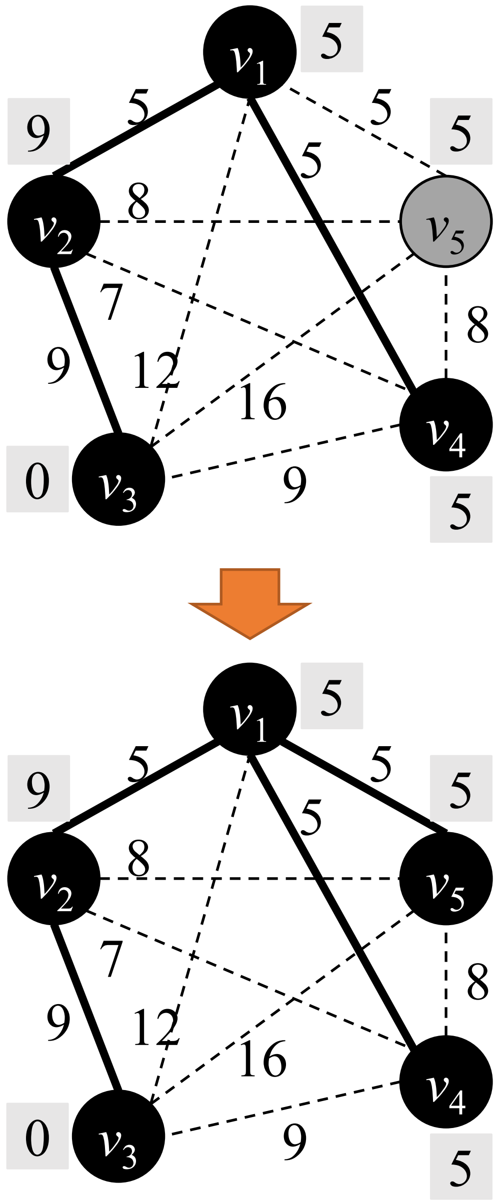
算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w\rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null;
集合 Q 初值为 V

$r \leftarrow V$ 中任意一个顶点;
$r.\mathrm{mw} \leftarrow 0$;
while $Q \neq \emptyset$ **do**
  $v \leftarrow \arg\min_{u \in Q} u.\mathrm{mw}$;
  $Q \leftarrow Q \setminus \{v\}$;
  if $v \neq r$ **then**
    输出 $((v.parent,\ v))$;
  foreach $(v, w) \in E$ **do**
    if $w \in Q$ 且 $w((v, w)) < w.\mathrm{mw}$ **then**
      $w.\mathrm{mw} \leftarrow w((v, w))$;
      $w.\mathrm{parent} \leftarrow v$;

黑色顶点：已增加到当前树中的顶点
粗实线：已增加到当前树中的边
灰色顶点：下轮while循环选择的顶点v



思考题6.9

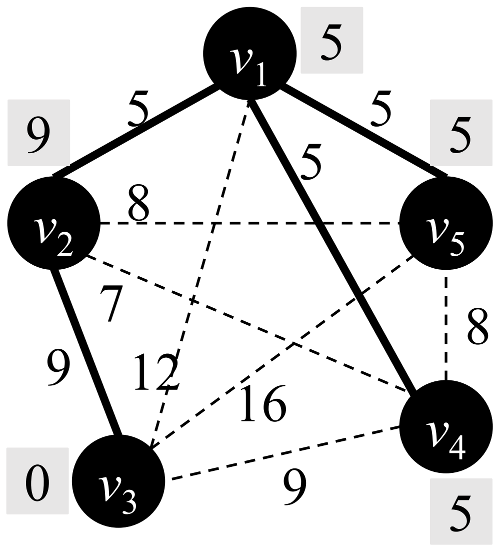
- 为什么普里姆算法输出的边组成一棵生成树?

算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null;
集合 Q 初值为 V

$r \leftarrow V$ 中任意一个顶点;
$r.\mathrm{mw} \leftarrow 0$;
while $Q \neq \emptyset$ **do**
  $v \leftarrow \arg\min_{u \in Q} u.\mathrm{mw}$;
  $Q \leftarrow Q \setminus \{v\}$;
  if $v \neq r$ **then**
    输出 $((v.parent, v))$;
  foreach $(v, w) \in E$ **do**
    if $w \in Q$ 且 $w((v, w)) < w.\mathrm{mw}$ **then**
      $w.\mathrm{mw} \leftarrow w((v, w))$;
      $w.\mathrm{parent} \leftarrow v$;



思考题6.9

- 为什么普里姆算法输出的边组成一棵生成树？
  - 连通
  - 边数 = 阶 – 1
  - 含G的所有顶点

算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；集合 Q 初值为 V

$r \leftarrow V$ 中任意一个顶点;

$r.\mathrm{mw} \leftarrow 0$;

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

    $v \leftarrow \arg\min_{u \in Q} u.\mathrm{mw}$;

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

    if $v \neq r$ **then**

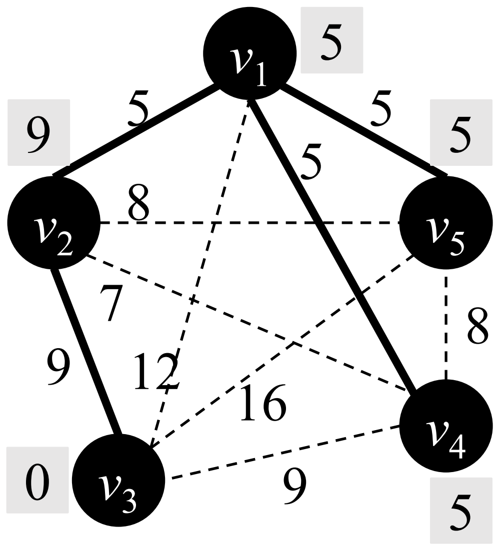
        输出 $((v.parent, v))$;

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

        if $w \in Q$ 且 $w((v, w)) < w.\mathrm{mw}$ **then**

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

            $w.\mathrm{parent} \leftarrow v$;



定理6.2

- 普里姆算法每轮while循环结束后，当前树是赋权图G的一棵最小生成树的子图。

算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；集合 Q 初值为 V

$r \leftarrow V$ 中任意一个顶点;

r.mw $\leftarrow 0$;

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

  $v \leftarrow \arg\min_{u \in Q} u.\mathrm{mw}$;

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

  if $v \neq r$ **then**

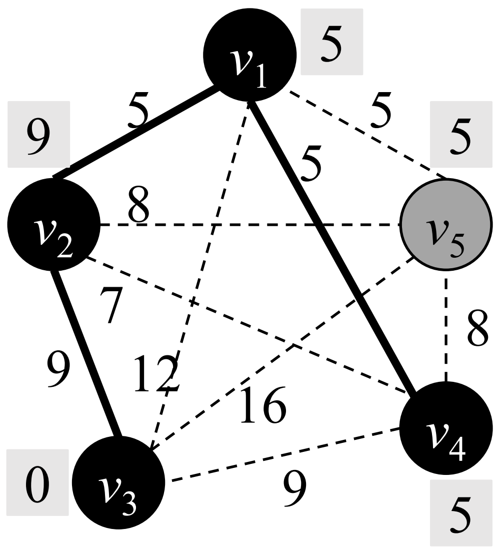
    输出 $((v.parent,\ v))$;

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

    if $w \in Q$ 且 $w((v, w)) < w.\mathrm{mw}$ **then**

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

      $w.\mathrm{parent} \leftarrow v$;



定理6.2

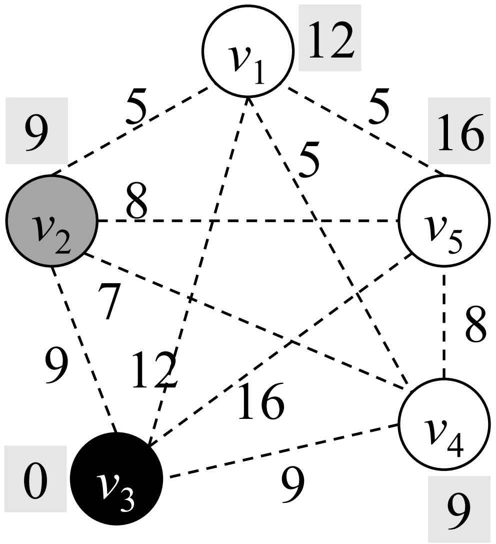
- 普里姆算法每轮while循环结束后，当前树是赋权图G的一棵最小生成树的子图。
  - 第1轮while循环结束后，当前树由顶点r组成，成立。

算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；集合 Q 初值为 V

```
r ← V 中任意一个顶点;
r.mw ← 0;
while Q ≠ ∅ do
    v ← arg min_{u∈Q} u.mw;
    Q ← Q \ {v};
    if v ≠ r then
        输出 ((v.parent, v));
    foreach (v, w) ∈ E do
        if w ∈ Q 且 w((v, w)) < w.mw then
            w.mw ← w((v, w));
            w.parent ← v;
```



定理6.2

- 普里姆算法每轮while循环结束后，当前树是赋权图G的一棵最小生成树的子图。
  - 若上轮while循环结束后成立，当前树T_i是赋权图G的最小生成树T^*的子图，则本轮while循环结束后：
    - 若当前树T_{i+1}是T^*的子图，则成立；

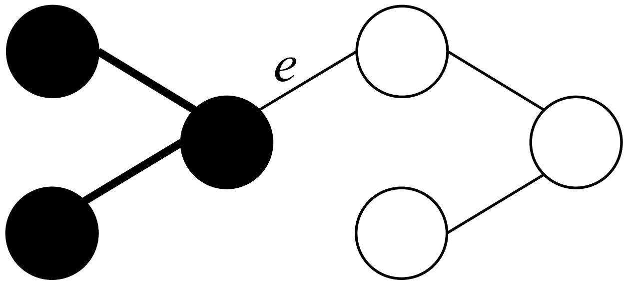
算法 6.2: 普里姆算法

输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null；集合 Q 初值为 V

$r \leftarrow V$ 中任意一个顶点;
r.mw $\leftarrow 0$;
while $Q \neq \emptyset$ **do**
  $v \leftarrow \arg\min_{u \in Q} u$.mw;
  $Q \leftarrow Q \setminus \{v\}$;
  if $v \neq r$ **then**
    输出 $((v.parent,\ v))$;
  foreach $(v, w) \in E$ **do**
    if $w \in Q$ 且 $w((v, w)) < w$.mw **then**
      w.mw $\leftarrow w((v, w))$;
      w.parent $\leftarrow v$;

实线：最小生成树T^*中的边
黑色顶点：已增加到当前树T_i中的顶点
粗实线：已增加到当前树T_i中的边



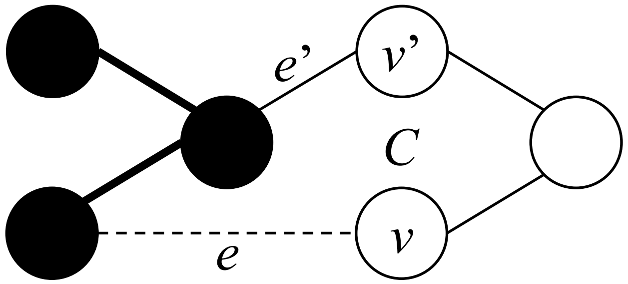
定理6.2

- 普里姆算法每轮while循环结束后，当前树是赋权图G的一棵最小生成树的子图。
  - 若上轮while循环结束后成立，当前树T_i是赋权图G的最小生成树T^*的子图，则本轮while循环结束后：
    - 若当前树T_{i+1}是T^*的子图，则成立；
    - 若T_{i+1}不是T^*的子图，则对于本轮while循环选择的顶点v和输出的边$e = (v.\text{parent}, v)$，T^*不含e，向T^*中增加e形成圈C，从C中删除T^*含而T_i不含的一条恰有一个端点在T_i中的边e'，将T^*变为G的另一棵生成树$T^\#$，将e'的不在T_i中的端点记作v'，根据普里姆算法，$w(e) = v.\text{mw} \le v'.\text{mw} \le w(e')$
    因此，$T^\#$也是G的最小生成树且T_{i+1}是$T^\#$的子图，成立。

算法 6.2: 普里姆算法

```
输入: 连通赋权图 G = ⟨V, E, w⟩
初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null;
      集合 Q 初值为 V
r ← V 中任意一个顶点;
r.mw ← 0;
while Q ≠ ∅ do
    v ← arg min_{u∈Q} u.mw;
    Q ← Q \ {v};
    if v ≠ r then
        输出 ((v.parent, v));
    foreach (v, w) ∈ E do
        if w ∈ Q 且 w((v, w)) < w.mw then
            w.mw ← w((v, w));
            w.parent ← v;
```

实线：最小生成树T^*中的边
黑色顶点：已增加到当前树T_i中的顶点
粗实线：已增加到当前树T_i中的边



普里姆算法

- 时间复杂度： $O(m+n\log n)$
  - 构造Q（基于斐波那契堆的优先队列）： $O(n)$
  - 每轮while循环：
    - 从Q中选择并删除顶点v： $O(\log n)$
  - 循环的轮数： $O(n)$
  - 所有循环更新mw属性值和parent属性值并维护Q： $O(m)$

算法 6.2: 普里姆算法

```
输入: 连通赋权图 G = ⟨V, E, w⟩
初值: 顶点集 V 中所有顶点的 mw 初值为 ∞，parent 初值为 null;
      集合 Q 初值为 V
r ← V 中任意一个顶点;
r.mw ← 0;
while Q ≠ ∅ do
    v ← arg min_{u∈Q} u.mw;
    Q ← Q \ {v};
    if v ≠ r then
        输出 ((v.parent, v));
    foreach (v, w) ∈ E do
        if w ∈ Q 且 w((v, w)) < w.mw then
            w.mw ← w((v, w));
            w.parent ← v;
```

接下来进入其它算法部分