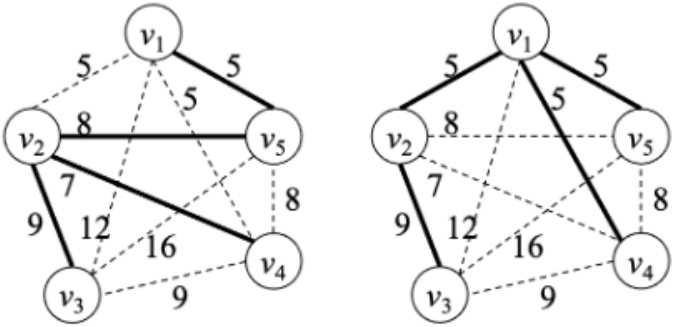
3-8 最小生成树

程龚

你能解释这些概念吗？

- 最小生成树

思考题 6.6 最小生成树唯一吗？

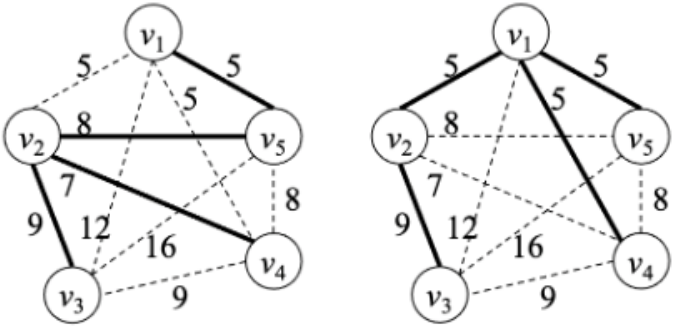


你能解释这些概念吗？

- 最小生成树

思考题 6.6 最小生成树唯一吗？

思考题 6.7 对于连通赋权图 $G = \langle V, E, w \rangle$，若边 $e \in E$ 在某个圈 C 中且 e 的权大于 C 中其它所有边的权，则 e 有可能出现在 G 的最小生成树中吗？



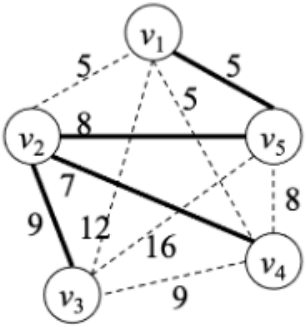
你能解释这些概念吗？

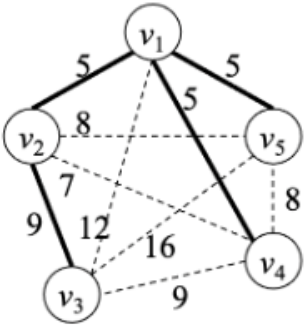
■ 最小生成树

思考题 6.6 最小生成树唯一吗？

思考题 6.7 对于连通赋权图 $G = \langle V, E, w \rangle$，若边 $e \in E$ 在某个圈 C 中且 e 的权大于 C 中其它所有边的权，则 e 有可能出现在 G 的最小生成树中吗？

思考题 6.8 对于连通赋权图 $G = \langle V, E, w \rangle$，若边 $e \in E$ 在某个极小边割集 S' 中且 e 的权小于 S' 中其它所有边的权，则 e 有可能不出现在 G 的最小生成树中吗？





你理解这个算法了吗？

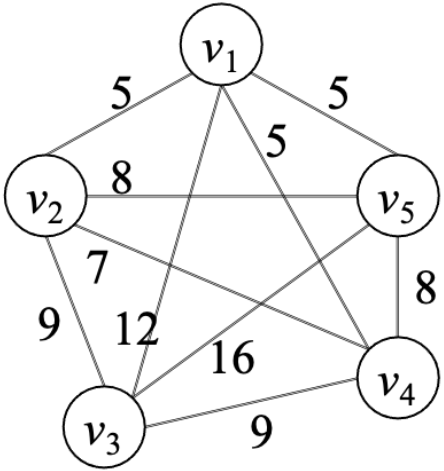
- Q的含义？
- mw的含义？

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;
```



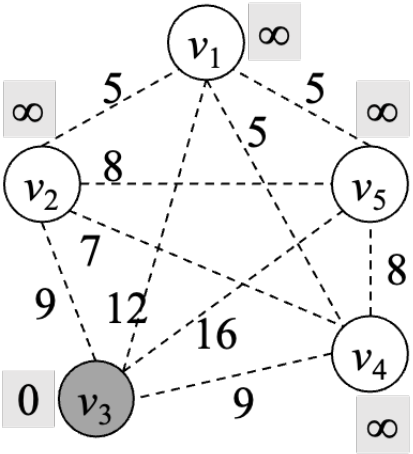
你理解这个算法了吗？

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;
```



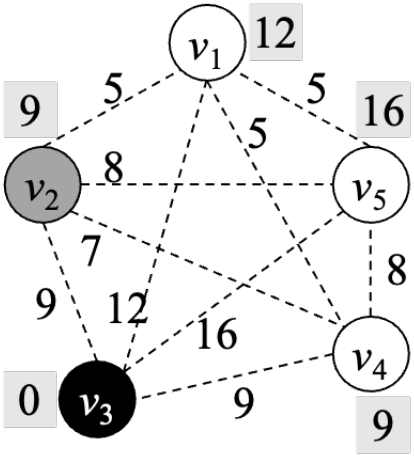
你理解这个算法了吗？

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;
```



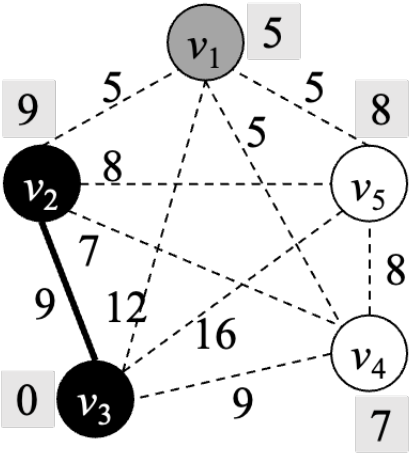
你理解这个算法了吗？

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;
```



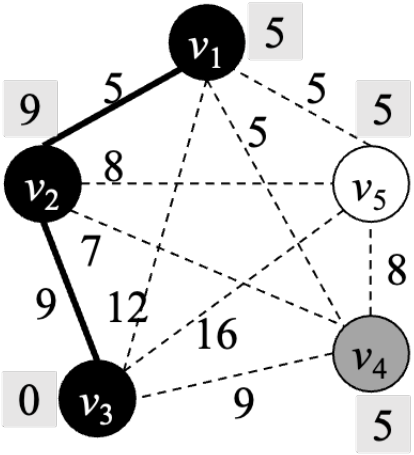
你理解这个算法了吗？

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;
```



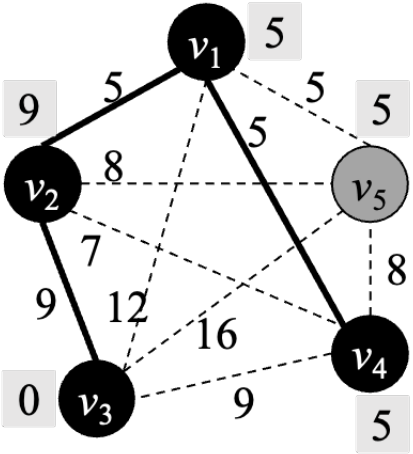
你理解这个算法了吗？

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;
```



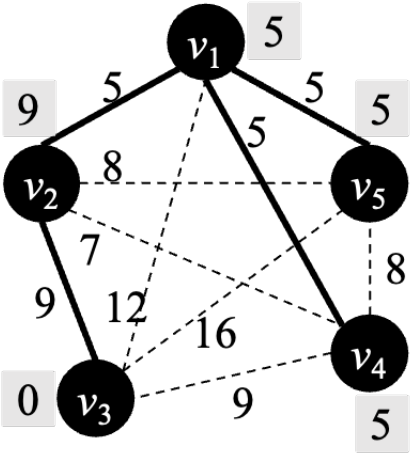
你理解这个算法了吗？

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;
```



你理解这个算法了吗？

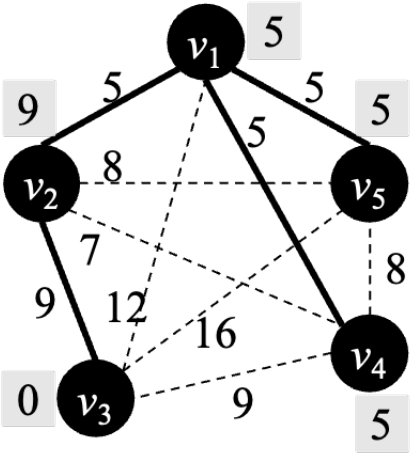
思考题 6.9 为什么普里姆算法输出的边组成一棵生成树？

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 ← 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;
```



你理解这个算法了吗？

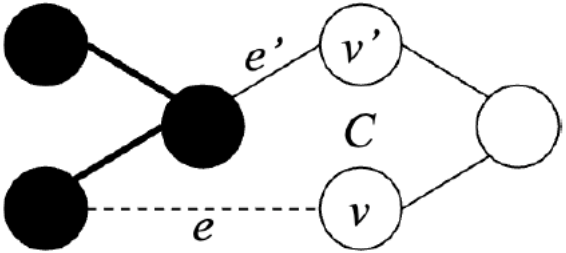
定理 6.2 普里姆算法每轮 while 循环结束后，当前树是赋权图 G 的一棵最小生成树的子图。

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 ← 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;
```



$$
\begin{aligned}
w(e) &= v.\mathrm{mw} \\
&\le v'.\mathrm{mw} \\
&\le w(e')\,.
\end{aligned}
$$

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，普里姆算法的时间复杂度为$O(m + n\log n)$。

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;
```

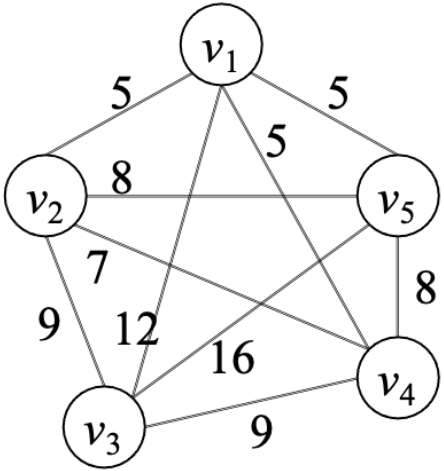
你理解这个算法了吗？

算法 6.3: 克拉斯克尔算法

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

初值: 边集 E 中所有边按权从小到大排序

```
foreach v ∈ V do
    建树 (v);
foreach (u, v) ∈ E do
    if 查树 (u) ≠ 查树 (v) then
        输出 ((u, v));
        并树 (u, v);
```



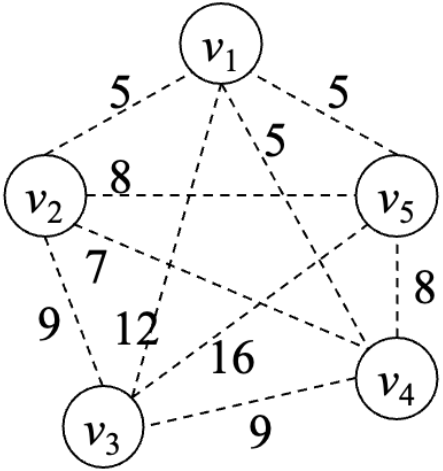
你理解这个算法了吗？

算法 6.3: 克拉斯克尔算法

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

初值: 边集 E 中所有边按权从小到大排序

```
foreach v ∈ V do
    建树（v）;
foreach (u, v) ∈ E do
    if 查树（u）≠ 查树（v）then
        输出（(u, v)）;
        并树（u, v）;
```



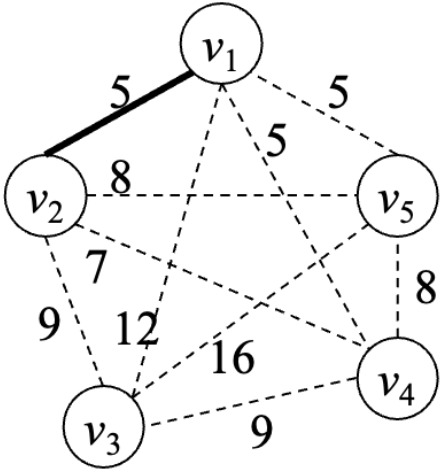
你理解这个算法了吗？

算法 6.3: 克拉斯克尔算法

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

初值: 边集 E 中所有边按权从小到大排序

```
foreach v ∈ V do
    建树（v）;
foreach (u, v) ∈ E do
    if 查树（u）≠ 查树（v）then
        输出（(u, v)）;
        并树（u, v）;
```



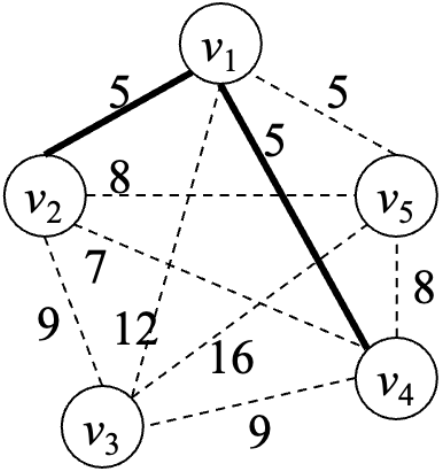
你理解这个算法了吗？

算法 6.3: 克拉斯克尔算法

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

初值: 边集 E 中所有边按权从小到大排序

```
foreach v ∈ V do
    建树（v）;
foreach (u, v) ∈ E do
    if 查树（u）≠ 查树（v）then
        输出（(u, v)）;
        并树（u, v）;
```



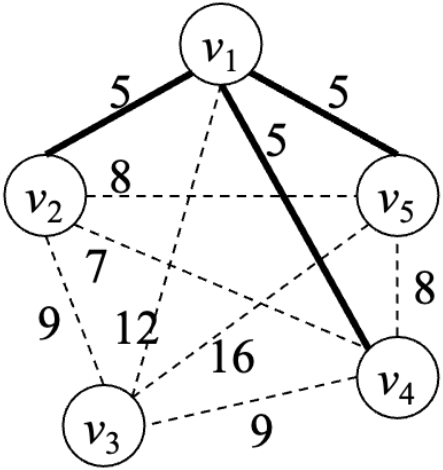
你理解这个算法了吗？

算法 6.3: 克拉斯克尔算法

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

初值: 边集 E 中所有边按权从小到大排序

```
foreach v ∈ V do
    建树 (v);
foreach (u, v) ∈ E do
    if 查树 (u) ≠ 查树 (v) then
        输出 ((u, v));
        并树 (u, v);
```



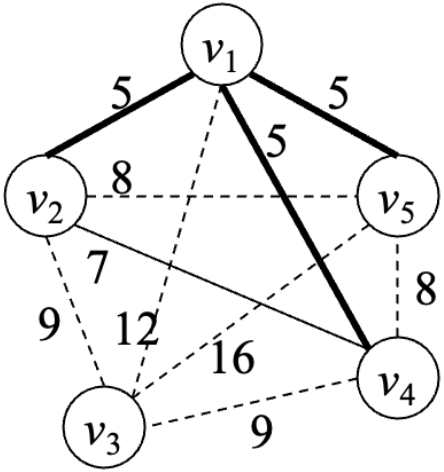
你理解这个算法了吗？

算法 6.3: 克拉斯克尔算法

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

初值: 边集 E 中所有边按权从小到大排序

```
foreach v ∈ V do
    建树 (v);
foreach (u, v) ∈ E do
    if 查树 (u) ≠ 查树 (v) then
        输出 ((u, v));
        并树 (u, v);
```



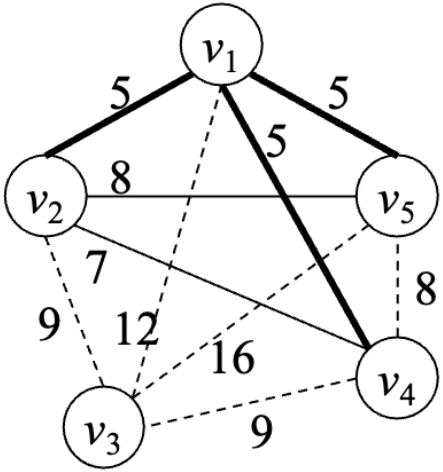
你理解这个算法了吗？

算法 6.3: 克拉斯克尔算法

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

初值: 边集 E 中所有边按权从小到大排序

```
foreach v ∈ V do
    建树（v）;
foreach (u, v) ∈ E do
    if 查树（u）≠ 查树（v）then
        输出（(u, v)）;
        并树（u, v）;
```



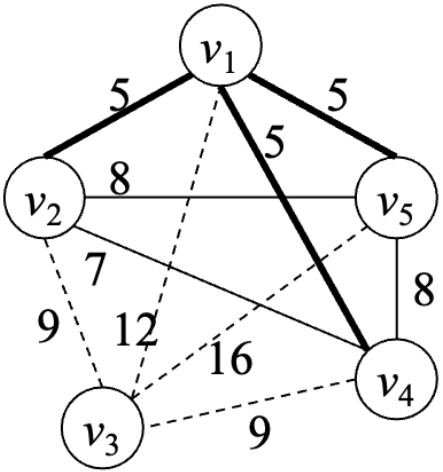
你理解这个算法了吗？

算法 6.3: 克拉斯克尔算法

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

初值: 边集 E 中所有边按权从小到大排序

```
foreach v ∈ V do
    建树（v）;
foreach (u, v) ∈ E do
    if 查树（u）≠ 查树（v）then
        输出（(u, v)）;
        并树（u, v）;
```



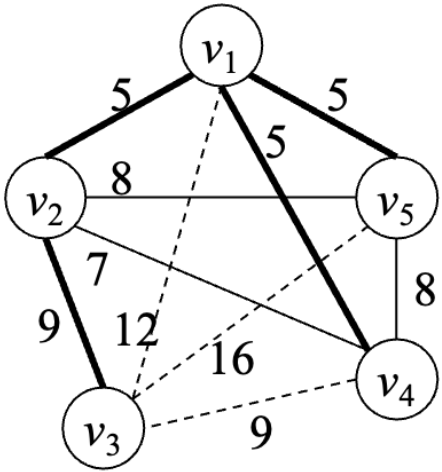
你理解这个算法了吗？

算法 6.3: 克拉斯克尔算法

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

初值: 边集 E 中所有边按权从小到大排序

```
foreach v ∈ V do
    建树（v）;
foreach (u, v) ∈ E do
    if 查树（u）≠ 查树（v）then
        输出（(u, v)）;
        并树（u, v）;
```



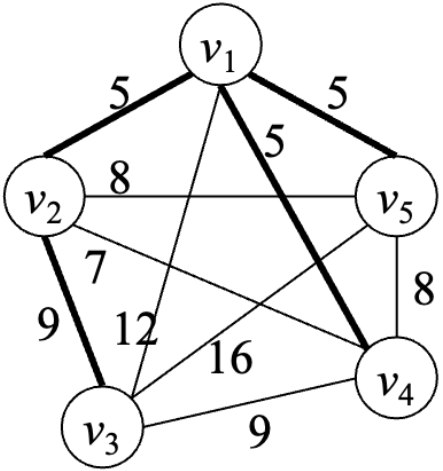
你理解这个算法了吗？

算法 6.3: 克拉斯克尔算法

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

初值: 边集 E 中所有边按权从小到大排序

```
foreach v ∈ V do
    建树（v）;
foreach (u, v) ∈ E do
    if 查树（u）≠ 查树（v）then
        输出（(u, v)）;
        并树（u, v）;
```



你理解这个算法了吗？

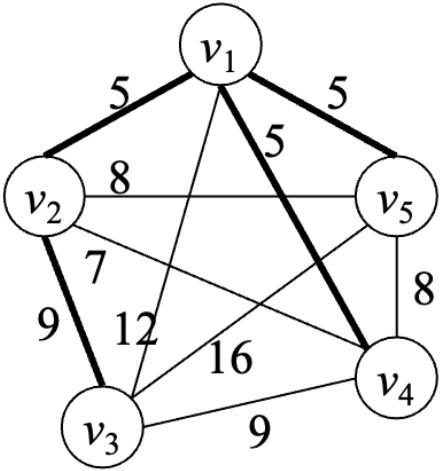
思考题 6.11 为什么克拉斯克尔算法输出的边组成一棵生成树？

算法 6.3: 克拉斯克尔算法

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

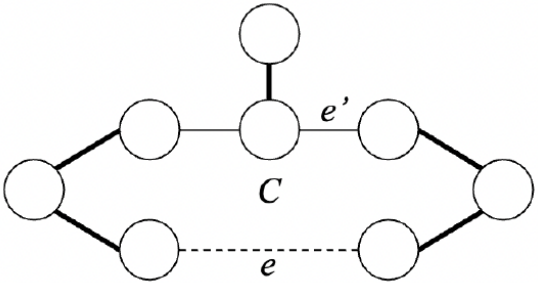
初值: 边集 E 中所有边按权从小到大排序

```
foreach v ∈ V do
    建树 (v);
foreach (u, v) ∈ E do
    if 查树 (u) ≠ 查树 (v) then
        输出 ((u, v));
        并树 (u, v);
```



你理解这个算法了吗？

定理 6.3 克拉斯克尔算法每轮 foreach 循环（第 3 行）开始前，当前森林是赋权图 G 的一棵最小生成树的子图。



$$w(e) \leq w(e')$$

算法 6.3: 克拉斯克尔算法

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

初值: 边集 E 中所有边按权从小到大排序

1 **foreach** $v \in V$ **do**
2 　建树（v）;
3 **foreach** $(u, v) \in E$ **do**
4 　**if** 查树（u）$\neq$ 查树（v）**then**
5 　　输出（(u, v)）;
6 　　并树（u, v）;

你理解这个算法的时间复杂度了吗？

对于阶为 n、边数为 m 的赋权图，克拉斯克尔算法对边集 E 中所有边排序的时间复杂度为 $O(m \log m)$；若采用按秩合并优化和路径压缩优化的并查集来存储当前森林中各棵树的顶点集，则所有建树、查树、并树的时间复杂度为 $O((n+m) \cdot \alpha(n))$，其中函数 α 是实际值可视作常数的反阿克曼函数。因此，该算法的时间复杂度为 $O(m \log m)$。

算法 6.3: 克拉斯克尔算法

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

初值: 边集 E 中所有边按权从小到大排序

```
foreach v ∈ V do
    建树（v）;
foreach (u, v) ∈ E do
    if 查树（u）≠ 查树（v）then
        输出（(u, v)）;
        并树（u, v）;
```

你理解这个算法的时间复杂度了吗？

对于阶为 n、边数为 m 的赋权图，克拉斯克尔算法对边集 E 中所有边排序的时间复杂度为 $O(m \log m)$；若采用按秩合并优化和路径压缩优化的并查集来存储当前森林中各棵树的顶点集，则所有建树、查树、并树的时间复杂度为 $O((n+m) \cdot \alpha(n))$，其中函数 α 是实际值可视作常数的反阿克曼函数。因此，该算法的时间复杂度为 $O(m \log m)$。

Suppose that all edge weights in a graph are integers in the range from 1 to $|V|$. How fast can you make Kruskal's algorithm run? What if the edge weights are integers in the range from 1 to W for some constant W?

算法 6.3: 克拉斯克尔算法

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

初值: 边集 E 中所有边按权从小到大排序

```
foreach v ∈ V do
    建树 (v);
foreach (u, v) ∈ E do
    if 查树 (u) ≠ 查树 (v) then
        输出 ((u, v));
        并树 (u, v);
```

你理解这个算法的时间复杂度了吗？

对于阶为 n、边数为 m 的赋权图，克拉斯克尔算法对边集 E 中所有边排序的时间复杂度为 $O(m \log m)$；若采用按秩合并优化和路径压缩优化的并查集来存储当前森林中各棵树的顶点集，则所有建树、查树、并树的时间复杂度为 $O((n+m) \cdot \alpha(n))$，其中函数 α 是实际值可视作常数的反阿克曼函数。因此，该算法的时间复杂度为 $O(m \log m)$。

Suppose that the edge weights in a graph are uniformly distributed over the half-open interval $[0, 1)$. Which algorithm, Kruskal's or Prim's, can you make run faster?

算法 6.3: 克拉斯克尔算法

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

初值: 边集 E 中所有边按权从小到大排序

```
foreach v ∈ V do
    建树 (v);
foreach (u, v) ∈ E do
    if 查树 (u) ≠ 查树 (v) then
        输出 ((u, v));
        并树 (u, v);
```

你能比较这两个算法的异同吗？

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;
```

算法 6.3: 克拉斯克尔算法

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

初值: 边集 E 中所有边按权从小到大排序

```
foreach v ∈ V do
    建树 (v);
foreach (u, v) ∈ E do
    if 查树 (u) ≠ 查树 (v) then
        输出 ((u, v));
        并树 (u, v);
```

你理解这个算法框架了吗？

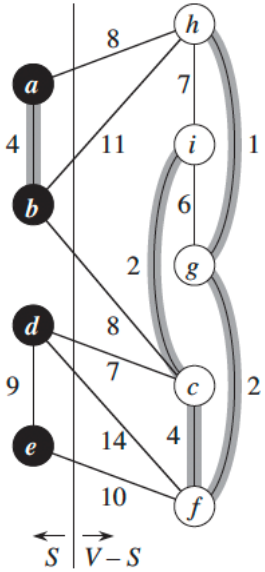
```
GENERIC-MST(G, w)
A = Ø
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```

an edge that we can add to A without violating the invariant:

Prior to each iteration, A is a subset of some minimum spanning tree.

你理解这个算法框架了吗？

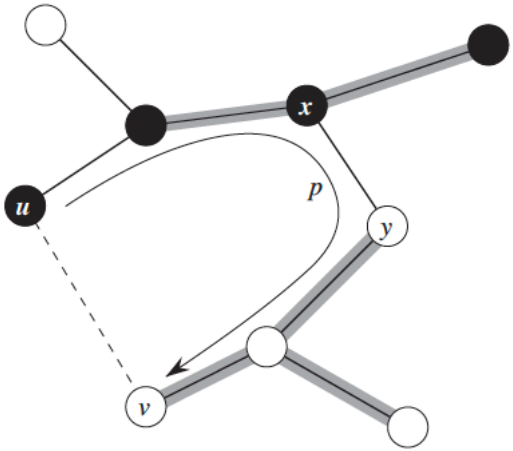
We first need some definitions. A ***cut*** $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V. Figure 23.2 illustrates this notion. We say that an edge $(u, v) \in E$ ***crosses*** the cut $(S, V - S)$ if one of its endpoints is in S and the other is in $V - S$. We say that a cut ***respects*** a set A of edges if no edge in A crosses the cut. An edge is a ***light edge*** crossing a cut if its weight is the minimum of any edge crossing the cut. Note that there can be more than one light edge crossing a cut in the case of ties. More generally, we say that an edge is a ***light edge*** satisfying a given property if its weight is the minimum of any edge satisfying the property.



你理解这个算法框架了吗？

Theorem 23.1

Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a light edge crossing $(S, V - S)$. Then, edge (u, v) is safe for A.



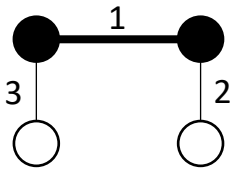
$$\begin{aligned} w(T') &= w(T) - w(x, y) + w(u, v) \\ &\leq w(T) . \end{aligned}$$

你理解这个算法框架了吗？

Professor Sabatier conjectures the following converse of Theorem 23.1. Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a safe edge for A crossing $(S, V - S)$. Then, (u, v) is a light edge for the cut. Show that the professor's conjecture is incorrect by giving a counterexample.

你理解这个算法框架了吗？

Professor Sabatier conjectures the following converse of Theorem 23.1. Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a safe edge for A crossing $(S, V - S)$. Then, (u, v) is a light edge for the cut. Show that the professor's conjecture is incorrect by giving a counterexample.



你理解这个算法框架了吗？

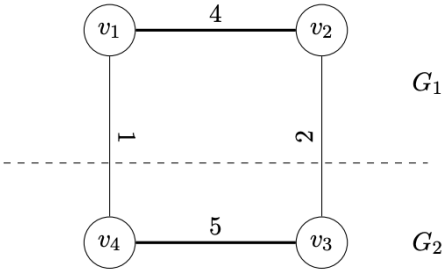
Professor Borden proposes a new divide-and-conquer algorithm for computing minimum spanning trees, which goes as follows. Given a graph $G = (V, E)$, partition the set V of vertices into two sets V_1 and V_2 such that $|V_1|$ and $|V_2|$ differ by at most 1. Let E_1 be the set of edges that are incident only on vertices in V_1, and let E_2 be the set of edges that are incident only on vertices in V_2. Recursively solve a minimum-spanning-tree problem on each of the two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Finally, select the minimum-weight edge in E that crosses the cut (V_1, V_2), and use this edge to unite the resulting two minimum spanning trees into a single spanning tree.

Either argue that the algorithm correctly computes a minimum spanning tree of G, or provide an example for which the algorithm fails.

你理解这个算法框架了吗？

Professor Borden proposes a new divide-and-conquer algorithm for computing minimum spanning trees, which goes as follows. Given a graph $G = (V, E)$, partition the set V of vertices into two sets V_1 and V_2 such that $|V_1|$ and $|V_2|$ differ by at most 1. Let E_1 be the set of edges that are incident only on vertices in V_1, and let E_2 be the set of edges that are incident only on vertices in V_2. Recursively solve a minimum-spanning-tree problem on each of the two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Finally, select the minimum-weight edge in E that crosses the cut (V_1, V_2), and use this edge to unite the resulting two minimum spanning trees into a single spanning tree.

Either argue that the algorithm correctly computes a minimum spanning tree of G, or provide an example for which the algorithm fails.

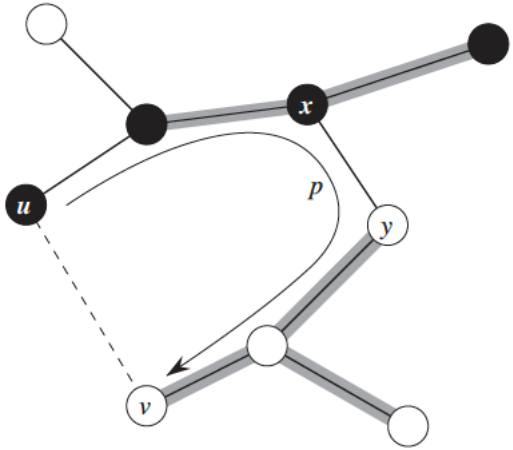


你理解这个算法框架了吗？

Show that if an edge (u, v) is contained in some minimum spanning tree, then it is a light edge crossing some cut of the graph.

你理解这个算法框架了吗？

Show that if an edge (u, v) is contained in some minimum spanning tree, then it is a light edge crossing some cut of the graph.



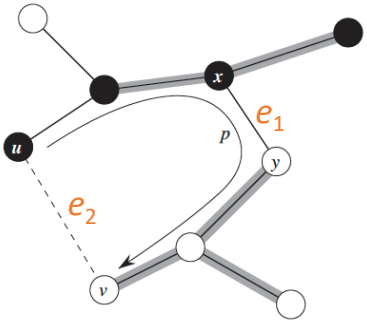
你理解这个算法框架了吗？

Show that a graph has a unique minimum spanning tree if, for every cut of the graph, there is a unique light edge crossing the cut. Show that the converse is not true by giving a counterexample.

你理解这个算法框架了吗？

Show that a graph has a unique minimum spanning tree if, for every cut of the graph, there is a unique light edge crossing the cut. Show that the converse is not true by giving a counterexample.

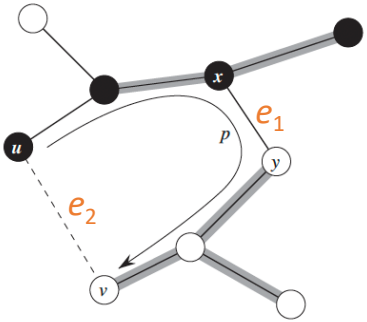
Solution: Suppose MST is not unique, i.e., there exist T_1 and T_2 where both of them are MST and they are not identical. Suppose $e_1 \in T_1$ but $e_1 \notin T_2$, if we remove e_1 from T_1, then we will have two trees with vertex sets V_1 and V_2. By problem 1 of HW #2, we know that e_1 is a minimum cost edge in the cut between V_1 and V_2. Now consider T_2, again problem 1 of HW #2, we know that T_2 contains an edge e_2 that is a minimum cost edge of the cut between V_1 and V_2. However, since $e_2 \neq e_1$ we must have: $c(e_1) = c(e_2)$ which is contradicting with the assumption that for every cut of the graph, the edge with the smallest cost across that cut is unique.

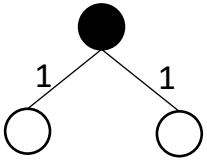


你理解这个算法框架了吗？

Show that a graph has a unique minimum spanning tree if, for every cut of the graph, there is a unique light edge crossing the cut. Show that the converse is not true by giving a counterexample.

Solution: Suppose MST is not unique, i.e., there exist T_1 and T_2 where both of them are MST and they are not identical. Suppose $e_1 \in T_1$ but $e_1 \notin T_2$, if we remove e_1 from T_1, then we will have two trees with vertex sets V_1 and V_2. By problem 1 of HW #2, we know that e_1 is a minimum cost edge in the cut between V_1 and V_2. Now consider T_2, again problem 1 of HW #2, we know that T_2 contains an edge e_2 that is a minimum cost edge of the cut between V_1 and V_2. However, since $e_2 \neq e_1$ we must have: $c(e_1) = c(e_2)$ which is contradicting with the assumption that for every cut of the graph, the edge with the smallest cost across that cut is unique.



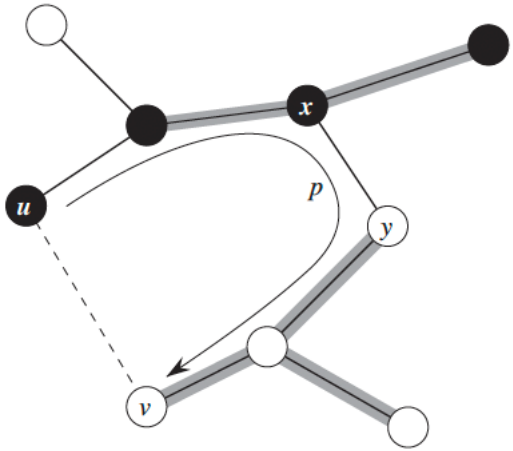


你理解这个算法框架了吗？

Let T be a minimum spanning tree of a graph G, and let L be the sorted list of the edge weights of T. Show that for any other minimum spanning tree T' of G, the list L is also the sorted list of edge weights of T'.

你理解这个算法框架了吗？

Let T be a minimum spanning tree of a graph G, and let L be the sorted list of the edge weights of T. Show that for any other minimum spanning tree T' of G, the list L is also the sorted list of edge weights of T'.



OT

- OT：最小生成树问题有很多相关问题，例如Steiner tree、k-MST、MBST等，请调研至少2种问题（其中至多1种来自上述例子，欧氏最小生成树问题等仅限制原始问题输入的问题不在调研范围内），讨论适用场景，形式化描述问题并概述现有解决方案。