第6章 赋权图

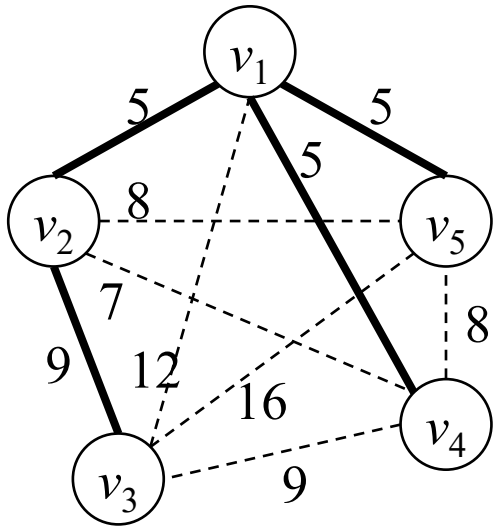
程龚
南京大学 计算机学院
gcheng@nju.edu.cn
http://ws.nju.edu.cn/~gcheng

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?

1. 普里姆算法
2. **克拉斯克尔算法**



克拉斯克尔算法

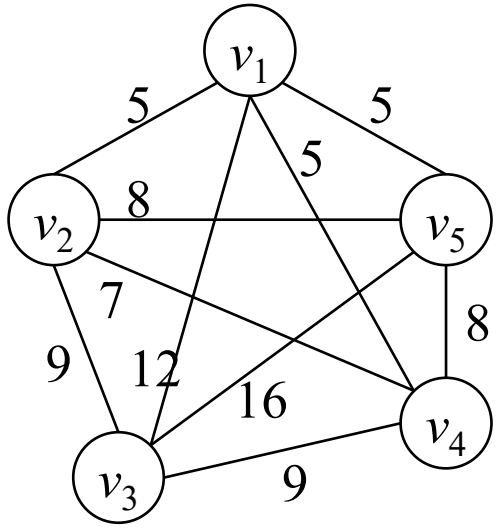
- 基本思路：逐步构造最小生成树，每步向当前森林增加一条边，并尽可能选择权小的边。

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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



克拉斯克尔算法

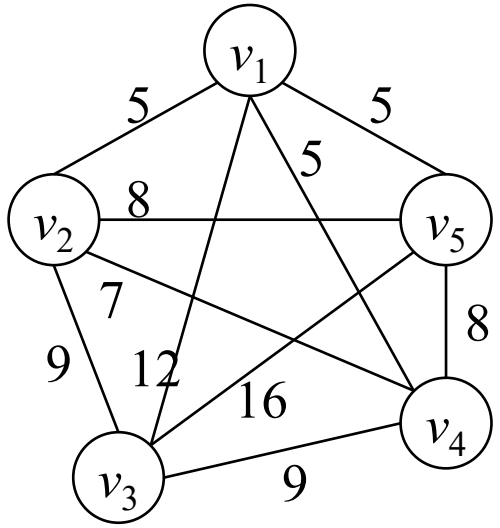
- 边集E中所有边按权从小到大排序。

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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



克拉斯克尔算法

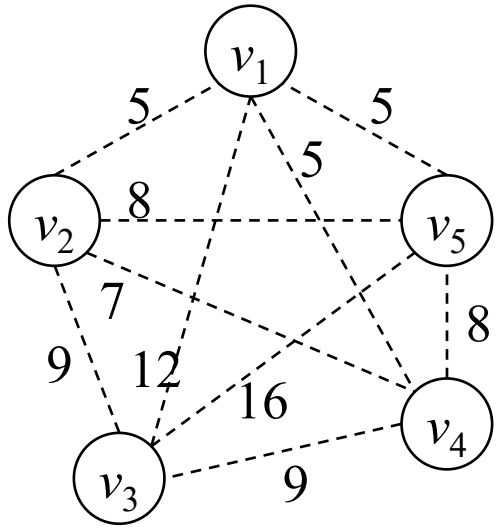
- 将顶点集V中每个顶点作为一棵平凡树形成一座森林。

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 ∈ V do
2 |  建树 (v);
3 foreach (u, v) ∈ E do
4 |  if 查树 (u) ≠ 查树 (v) then
5 |  |  输出 ((u, v));
6 |  |  并树 (u, v);
```



克拉斯克尔算法

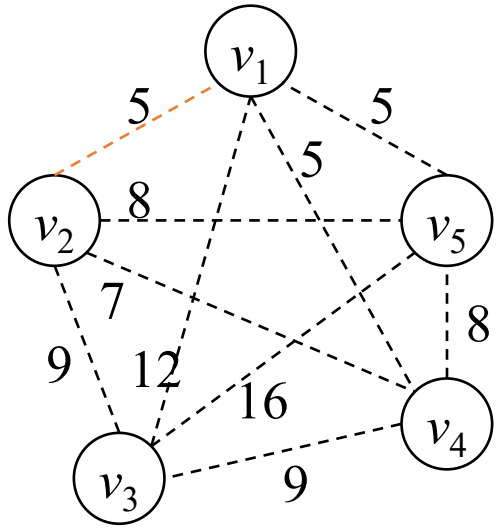
- 每轮foreach循环从E中选择一条边(u, v)：

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

```
1 foreach v ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



克拉斯克尔算法

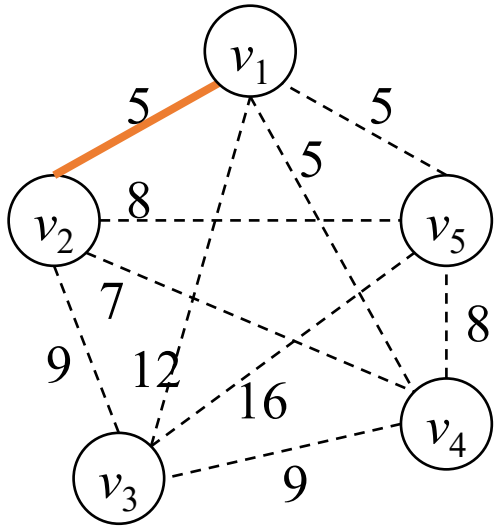
- 每轮foreach循环从E中选择一条边(u, v)：
  - 若端点u和v在不同的树中，则将(u, v)增加到当前森林中，u和v所在的两棵树被拼接为一棵

算法 6.3: 克拉斯克尔算法

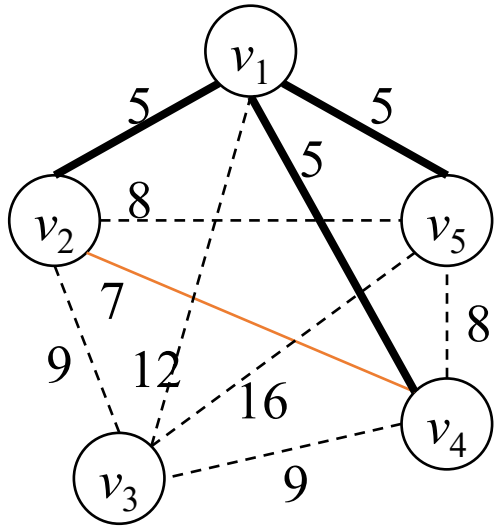
输入: 连通赋权图 $G = \langle V, E, w \rangle$

初值: 边集 E 中所有边按权从小到大排序

```
1 foreach v ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



u和v在不同的树中



u和v在相同的树中

克拉斯克尔算法

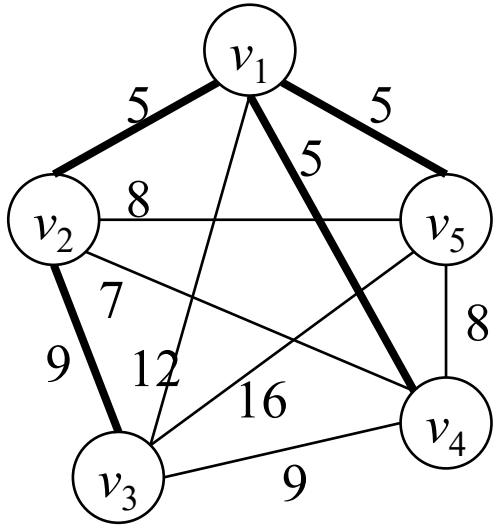
- 算法运行结束时，输出的边组成一棵最小生成树。

算法 6.3: 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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



克拉斯克尔算法

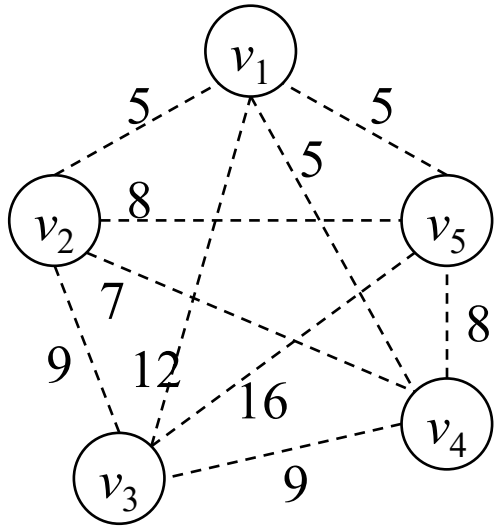
- 例如：从5棵平凡树开始

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 ∈ V do
2 |  建树 (v);
3 foreach (u, v) ∈ E do
4 |  if 查树 (u) ≠ 查树 (v) then
5 |  |  输出 ((u, v));
6 |  |  并树 (u, v);
```



克拉斯克尔算法

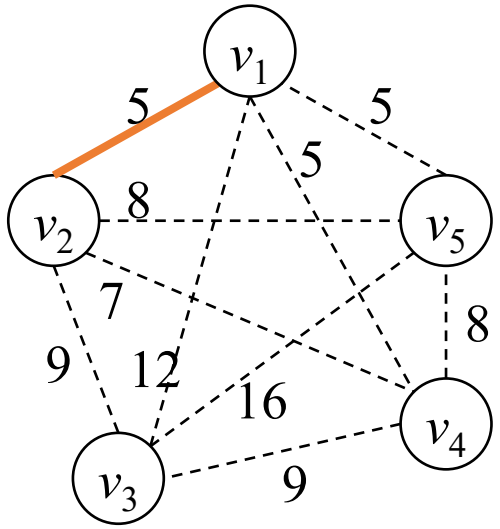
- 选择(v_1, v_2)
- 增加到当前森林中
- v_1和v_2所在的两棵树被拼接为一棵

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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



克拉斯克尔算法

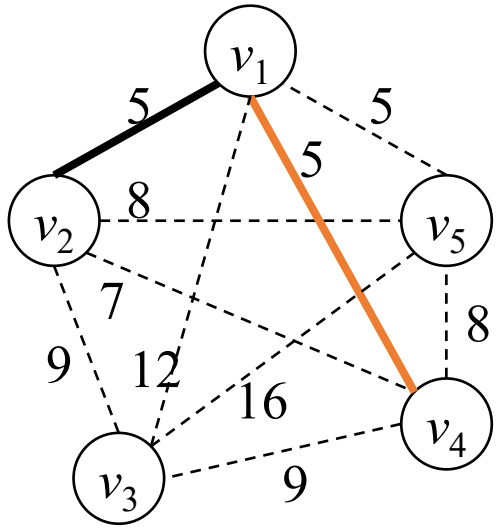
- 选择(v_1, v_4)
- 增加到当前森林中
- v_1和v_4所在的两棵树被拼接为一棵

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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



克拉斯克尔算法

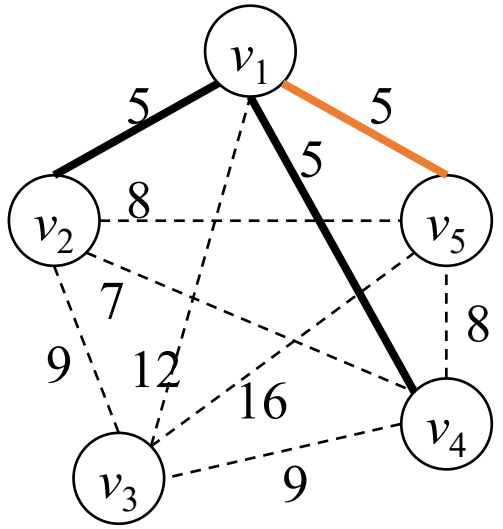
- 选择(v_1, v_5)
- 增加到当前森林中
- v_1和v_5所在的两棵树被拼接为一棵

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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



克拉斯克尔算法

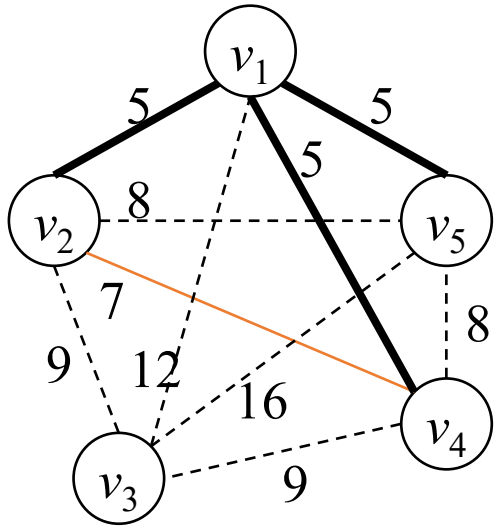
- 选择(v_2, v_4)
- 不增加到当前森林中

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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



克拉斯克尔算法

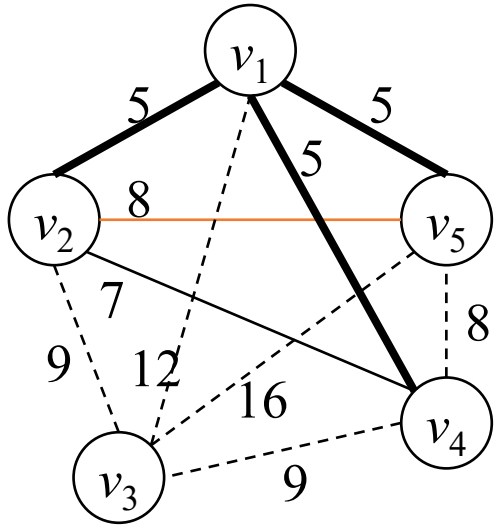
- 选择(v_2, v_5)
- 不增加到当前森林中

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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



克拉斯克尔算法

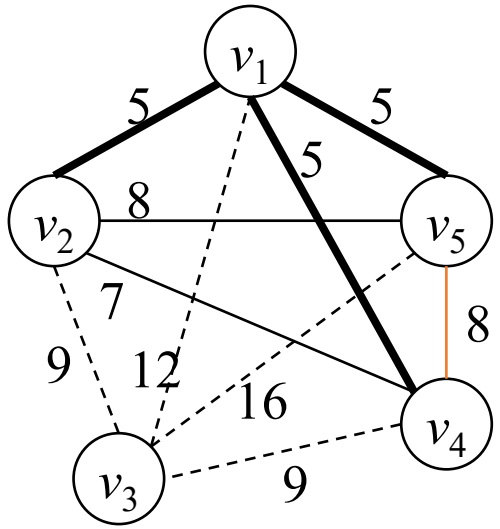
- 选择(v_4, v_5)
- 不增加到当前森林中

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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



克拉斯克尔算法

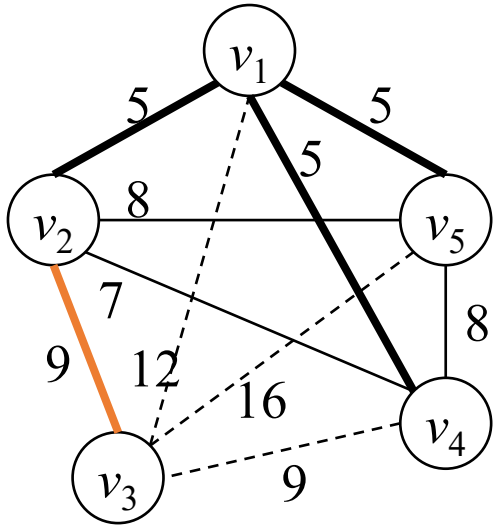
- 选择(v_2, v_3)
- 增加到当前森林中
- v_2和v_3所在的两棵树被拼接为一棵

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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



克拉斯克尔算法

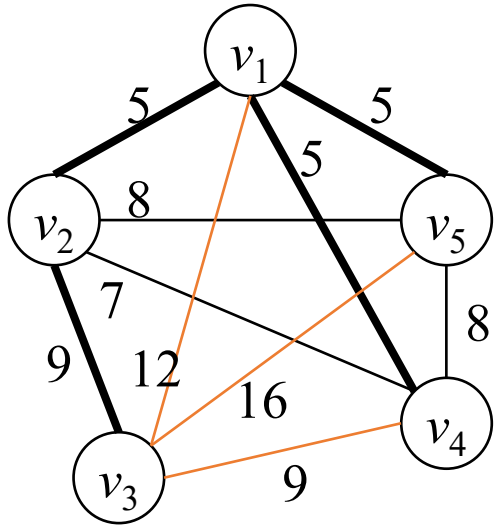
- 依次选择(v_3, v_4)、(v_1, v_3)、(v_3, v_5)
- 不增加到当前森林中

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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



思考题6.11

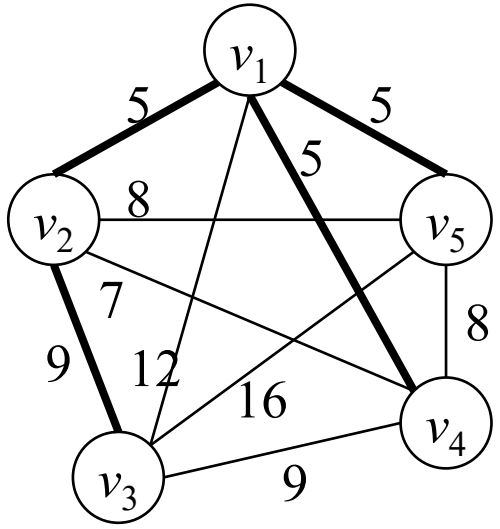
- 为什么克拉斯克尔算法输出的边组成一棵生成树?

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

```
1 foreach v ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



思考题6.11

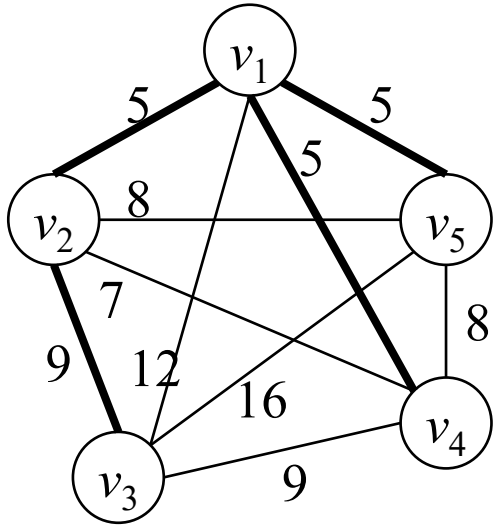
- 为什么克拉斯克尔算法输出的边组成一棵生成树?
  - 连通
  - 不含圈
  - 含G的所有顶点

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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



定理6.3

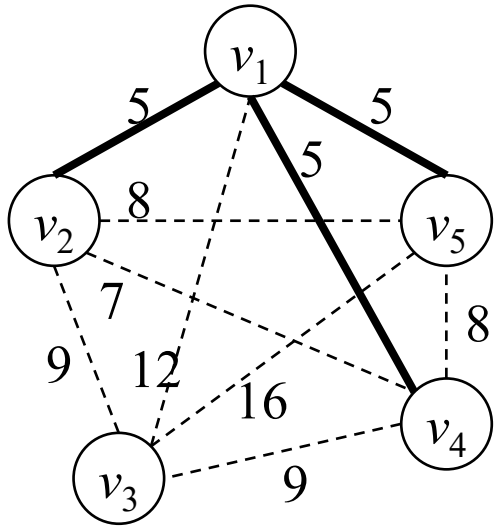
- 克拉斯克尔算法每轮foreach循环（第3行）开始前，当前森林是赋权图G的一棵最小生成树的子图。

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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



定理6.3

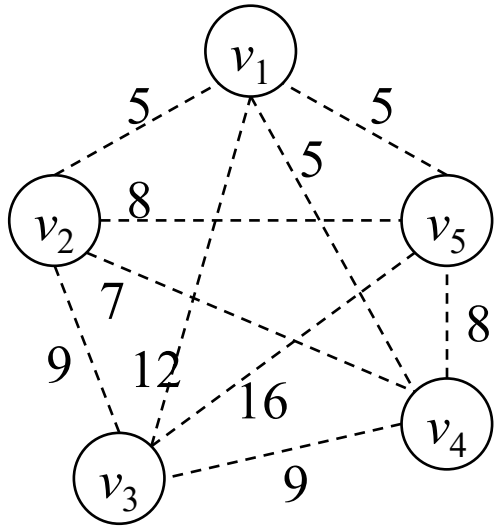
- 克拉斯克尔算法每轮foreach循环（第3行）开始前，
  当前森林是赋权图G的一棵最小生成树的子图。
  - 第1轮foreach循环开始前，当前森林由顶点集V中所有顶点组成，成立。

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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```



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。
  - 若本轮foreach循环条件判定前成立，当前森林F_i是赋权图G的最小生成树T^*的子图，
    则下轮foreach循环条件判定前：
    - 若当前森林F_{i+1}是T^*的子图，则成立；

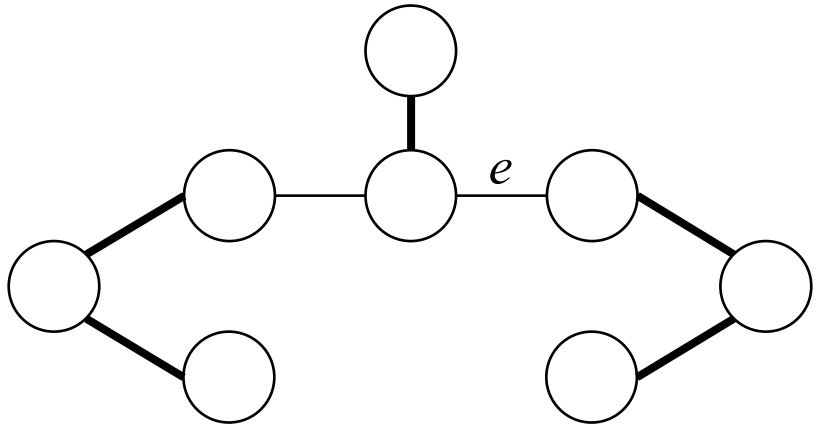
算法 6.3: 克拉斯克尔算法

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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```

实线：最小生成树T^*中的边
粗实线：当前森林F_i中的边



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。
  - 若本轮foreach循环条件判定前成立，当前森林F_i是赋权图G的最小生成树T^*的子图，则下轮foreach循环条件判定前：
    - 若当前森林F_{i+1}是T^*的子图，则成立；
    - 若F_{i+1}不是T^*的子图，则对于本轮foreach循环选择的边$e=(u,v)$，T^*不含e，向T^*中增加e形成圈C，从C中删除T^*含而F_i不含的一条边e'，将T^*变为G的另一棵生成树$T^\#$，根据克拉斯克尔算法，e'是未被选择的边，否则，选择e'时应将这条两个端点在不同的树中的边增加到F_i中，因此，$w(e) \le w(e')$，$T^\#$也是G的最小生成树且F_{i+1}是$T^\#$的子图，成立。

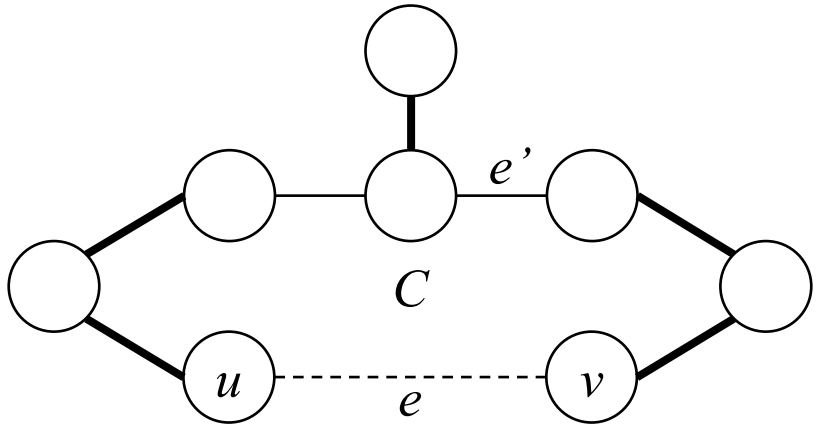
算法 6.3: 克拉斯克尔算法

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 ∈ V do
2 |   建树 (v);
3 foreach (u,v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u,v));
6 |   |   并树 (u, v);
```

实线：最小生成树T^*中的边
粗实线：当前森林F_i中的边



克拉斯克尔算法

- 时间复杂度：$O(m\log m)$
  - 对边集E中所有边排序：$O(m\log m)$
  - 所有建树、查树、并树（按秩合并优化和路径压缩优化的并查集）：$O((n+m)\,\alpha(n))$

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 ∈ V do
2 |   建树 (v);
3 foreach (u, v) ∈ E do
4 |   if 查树 (u) ≠ 查树 (v) then
5 |   |   输出 ((u, v));
6 |   |   并树 (u, v);
```

请认真完成课后练习