第6章 赋权图

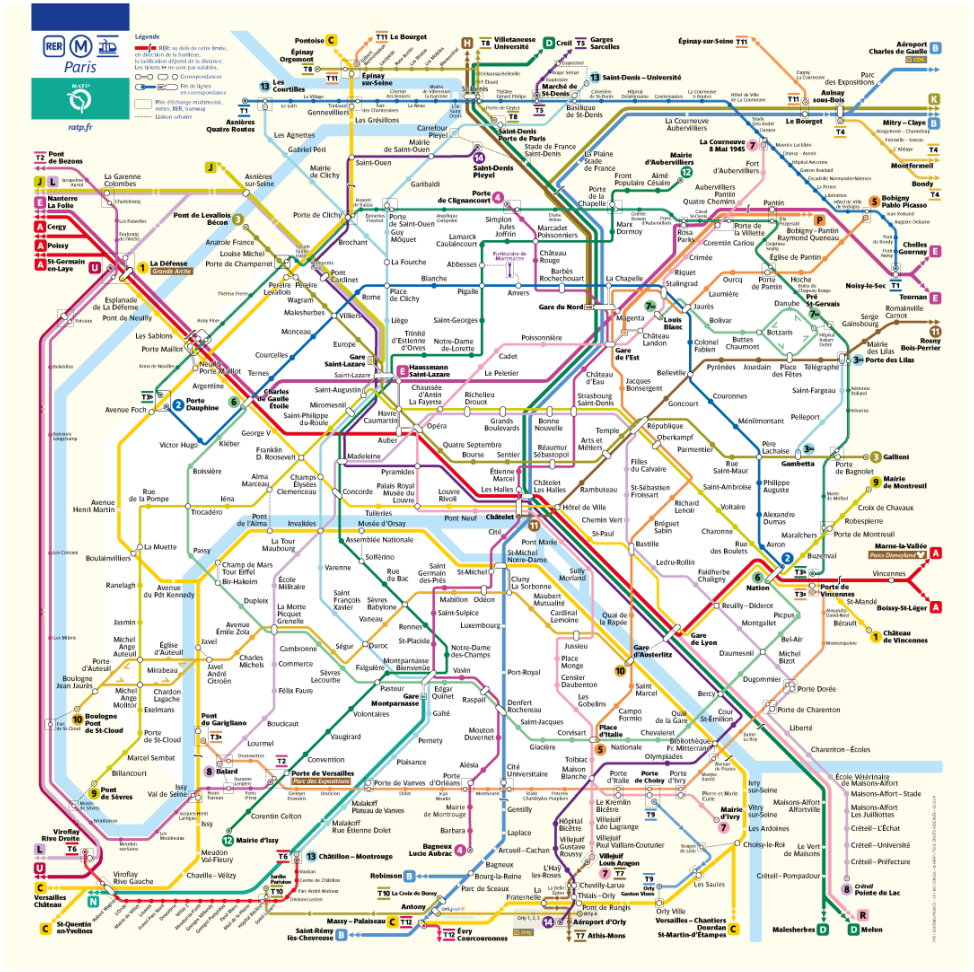
程龚
南京大学 计算机学院
gcheng@nju.edu.cn
http://ws.nju.edu.cn/~gcheng

本章内容

- 第6.1节 赋权图和距离
  - 第6.1.1节 理论
  - **第6.1.2节 算法**
- 第6.2节 最小生成树
- 第6.3节 赋权欧拉图
- 第6.4节 赋权哈密尔顿图

在赋权图中，如何计算两个顶点间的距离？
如何计算一个顶点和赋权图中所有顶点间的距离？

- 至少需要坐多远 或 多久？



https://www.ratp.fr/plan-de-ligne/img/metro/Plan-Metro.1722015166.png

戴克斯特拉算法

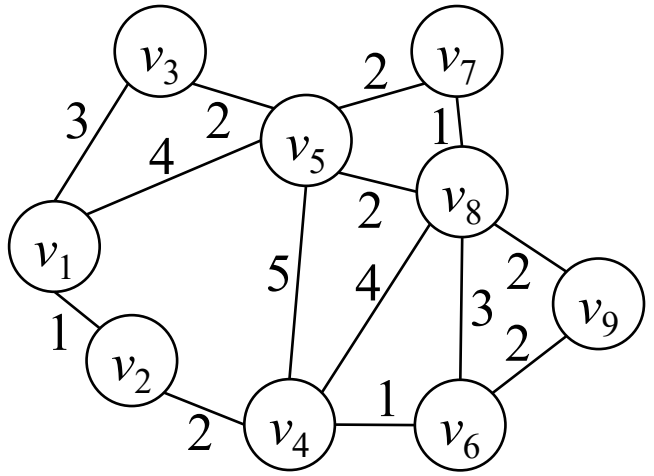
- 基本思路：逐步计算一个顶点和其它所有顶点间的距离，每步按从近到远的顺序计算一个顶点。

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

```
1 u.d ← 0;
2 while Q ≠ ∅ do
3 |   v ← arg min_{w∈Q} w.d;
4 |   Q ← Q \ {v};
5 |   foreach (v, w) ∈ E do
6 |   |   if w.d > v.d + w((v, w)) then
7 |   |   |   w.d ← v.d + w((v, w));
```



戴克斯特拉算法

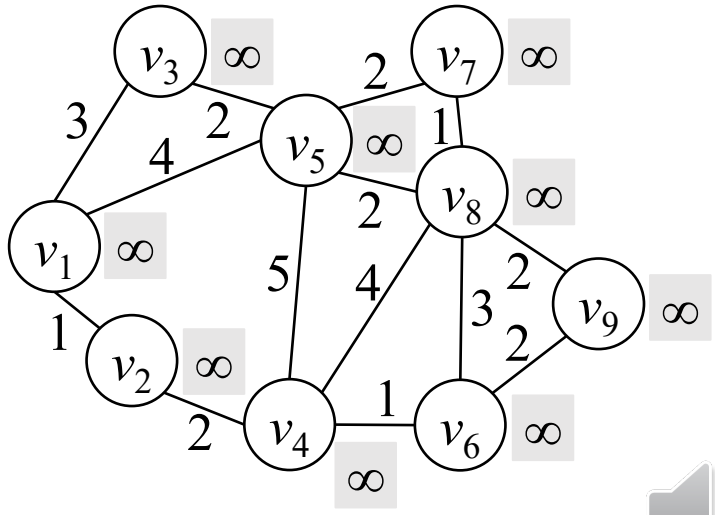
- 每个顶点的d属性：实数型变量，表示该顶点和指定顶点u间当前最短路的长度，初值为∞

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

```
1 u.d ← 0;
2 while Q ≠ ∅ do
3 |   v ← arg min_{w∈Q} w.d;
4 |   Q ← Q \ {v};
5 |   foreach (v, w) ∈ E do
6 |   |   if w.d > v.d + w((v, w)) then
7 |   |   |   w.d ← v.d + w((v, w));
```



戴克斯特拉算法

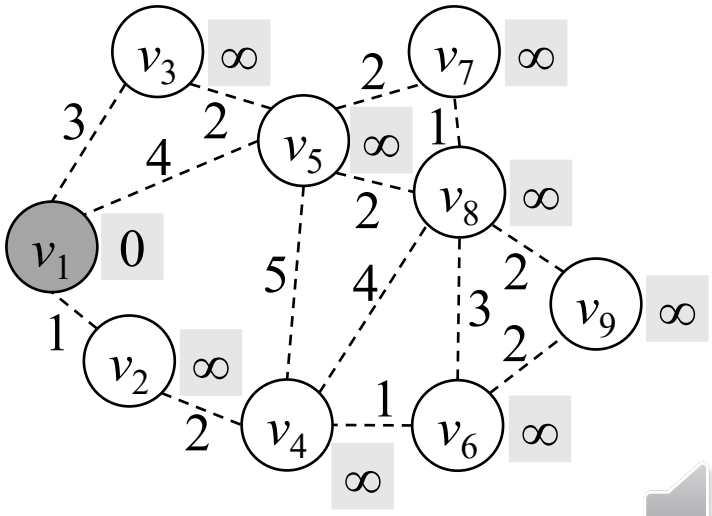
- 从u开始，每轮while循环从初值为V的集合Q中选择一个顶点v删除，直至Q为空：

算法 6.1: 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

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



戴克斯特拉算法

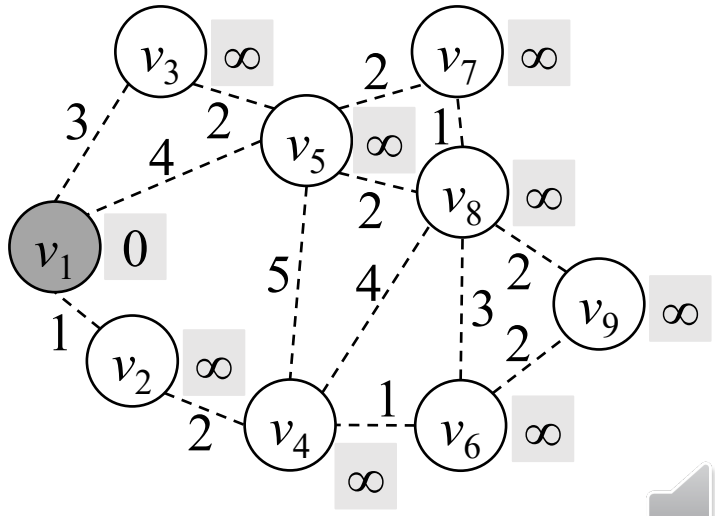
- 从u开始，每轮while循环从初值为V的集合Q中选择一个顶点v删除，直至Q为空：
  - v是Q中d属性值最小的顶点

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

```
1 u.d ← 0;
2 while Q ≠ ∅ do
3 |   v ← arg min_{w∈Q} w.d;
4 |   Q ← Q \ {v};
5 |   foreach (v, w) ∈ E do
6 |   |   if w.d > v.d + w((v, w)) then
7 |   |   |   w.d ← v.d + w((v, w));
```



戴克斯特拉算法

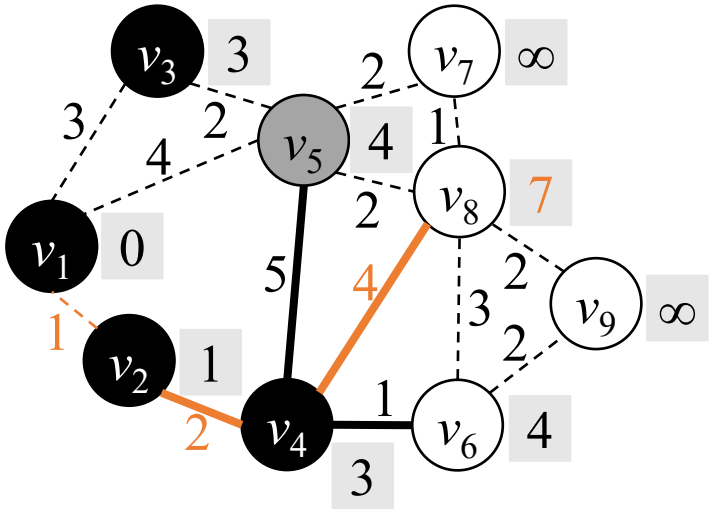
- 从u开始，每轮while循环从初值为V的集合Q中选择一个顶点v删除，直至Q为空：
  - v是Q中d属性值最小的顶点
  - 对于v的每个邻点w，若经过边(v, w)的u-w路比之前的最短u-w路更短，则更新w的d属性值

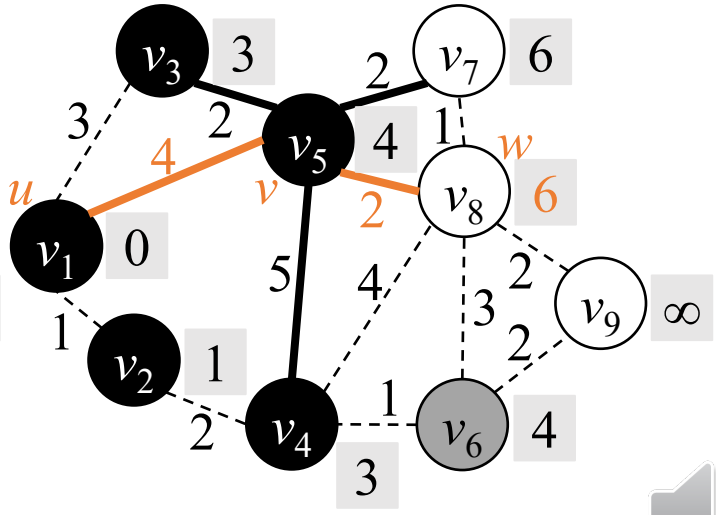
算法 6.1: 戴克斯特拉算法

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

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





戴克斯特拉算法

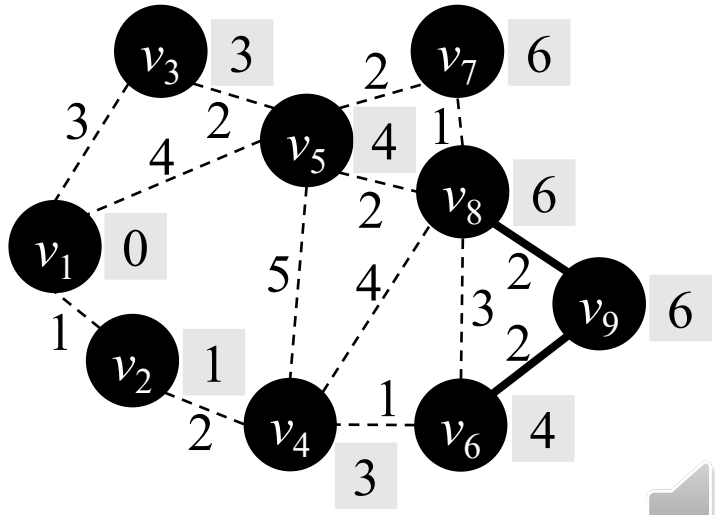
- 算法运行结束时，和u连通的所有顶点的d属性值为其和u间的距离。

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

```
1 u.d ← 0;
2 while Q ≠ ∅ do
3 |   v ← arg min_{w∈Q} w.d;
4 |   Q ← Q \ {v};
5 |   foreach (v, w) ∈ E do
6 |   |   if w.d > v.d + w((v, w)) then
7 |   |   |   w.d ← v.d + w((v, w));
```



戴克斯特拉算法

- 例如：从v_1开始，下轮while循环删除v_1

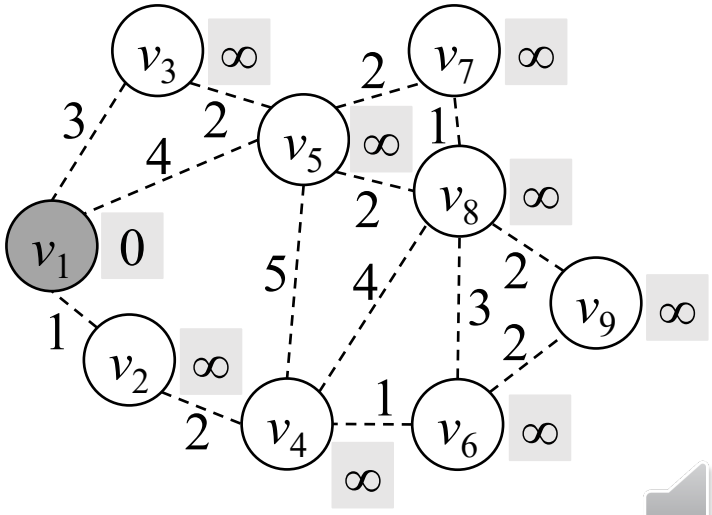
算法 6.1: 戴克斯特拉算法

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

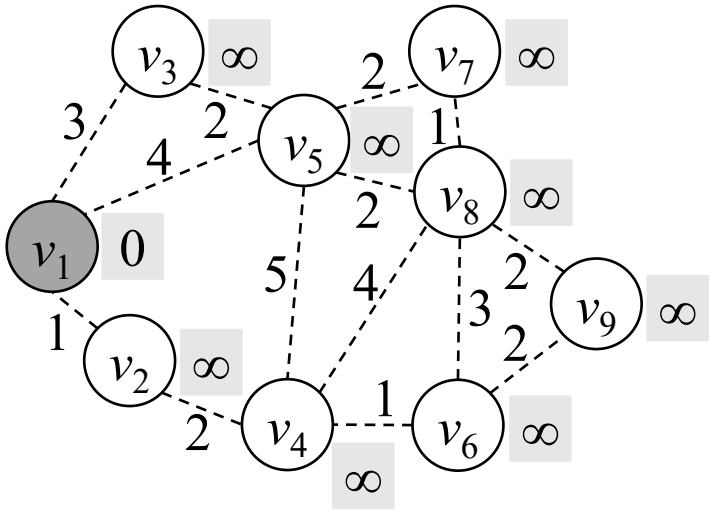
```
1 u.d ← 0;
2 while Q ≠ ∅ do
3 |   v ← arg min_{w∈Q} w.d;
4 |   Q ← Q \ {v};
5 |   foreach (v, w) ∈ E do
6 |   |   if w.d > v.d + w((v, w)) then
7 |   |   |   w.d ← v.d + w((v, w));
```

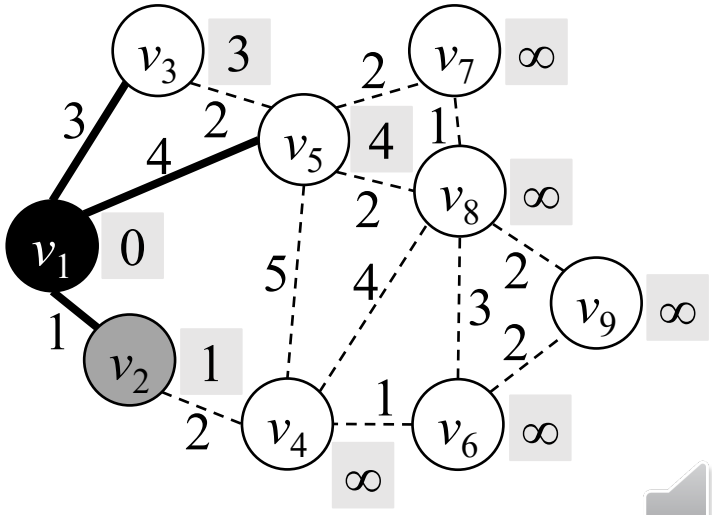
数字：d属性值
灰色顶点：下轮while循环删除的顶点v



戴克斯特拉算法

- 从Q中删除v_1
- 更新v_2、v_3、v_5的d属性值
- 下轮while循环删除v_2





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

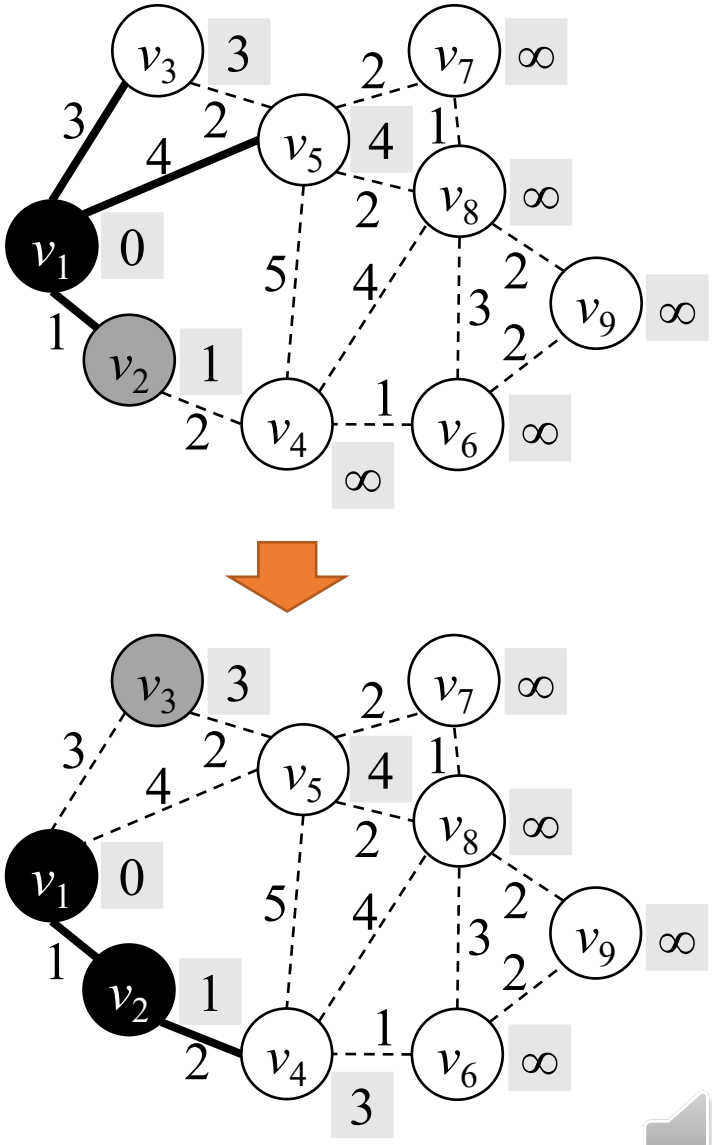
初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

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

数字：d属性值
灰色顶点：下轮while循环删除的顶点v
黑色顶点：已从Q中删除的顶点
粗实线：顶点v关联的边

戴克斯特拉算法

- 从Q中删除v_2
- 更新v_4的d属性值
- 下轮while循环删除v_3



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

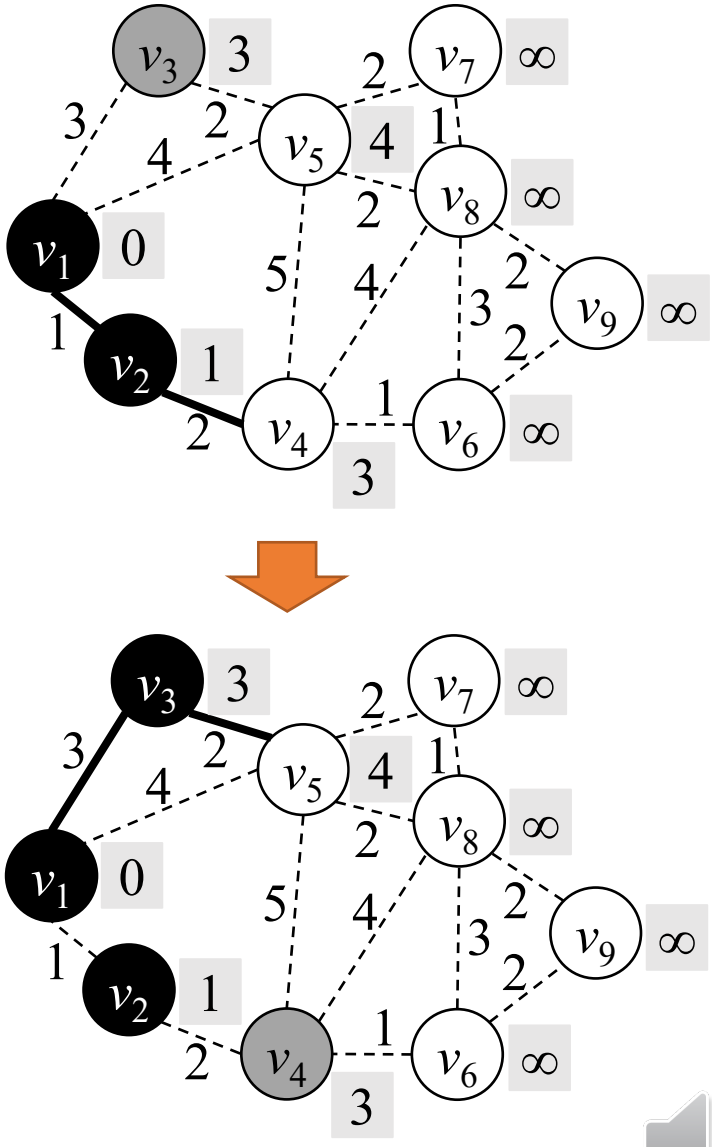
初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

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

数字：d属性值
灰色顶点：下轮while循环删除的顶点v
黑色顶点：已从Q中删除的顶点
粗实线：顶点v关联的边

戴克斯特拉算法

- 从Q中删除v_3
- 下轮while循环删除v_4



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

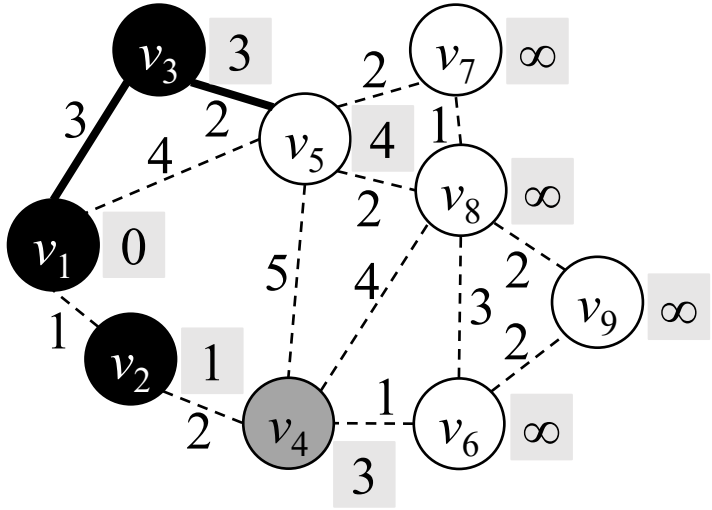
初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

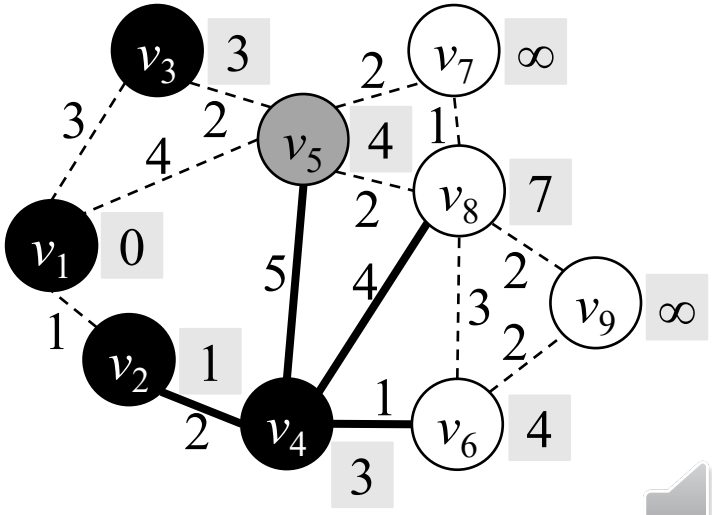
```
1 u.d ← 0;
2 while Q ≠ ∅ do
3     v ← arg min_{w∈Q} w.d;
4     Q ← Q \ {v};
5     foreach (v, w) ∈ E do
6         if w.d > v.d + w((v, w)) then
7             w.d ← v.d + w((v, w));
```

数字：d属性值
灰色顶点：下轮while循环删除的顶点v
黑色顶点：已从Q中删除的顶点
粗实线：顶点v关联的边

戴克斯特拉算法

- 从Q中删除v_4
- 更新v_6、v_8的d属性值
- 下轮while循环删除v_5





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

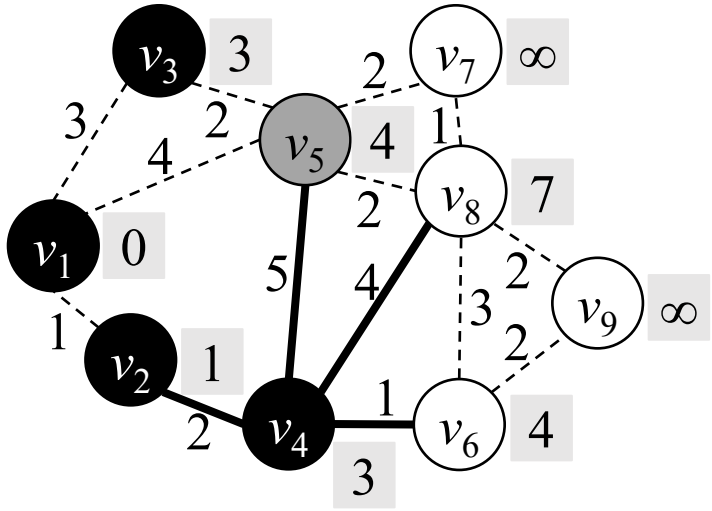
初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

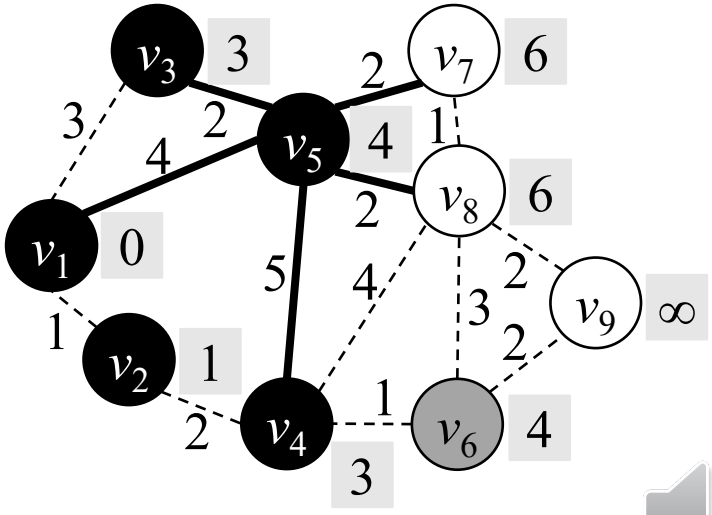
```
1 u.d ← 0;
2 while Q ≠ ∅ do
3     v ← arg min_{w∈Q} w.d;
4     Q ← Q \ {v};
5     foreach (v, w) ∈ E do
6         if w.d > v.d + w((v, w)) then
7             w.d ← v.d + w((v, w));
```

数字：d属性值
灰色顶点：下轮while循环删除的顶点v
黑色顶点：已从Q中删除的顶点
粗实线：顶点v关联的边

戴克斯特拉算法

- 从Q中删除v_5
- 更新v_7、v_8的d属性值
- 下轮while循环删除v_6





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

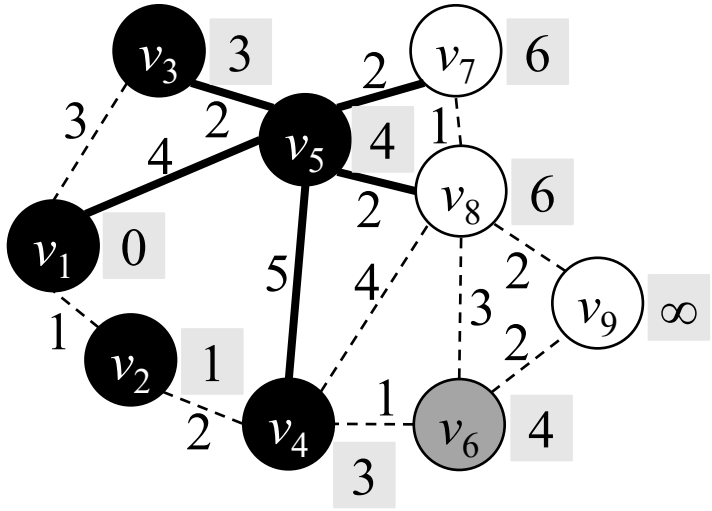
初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

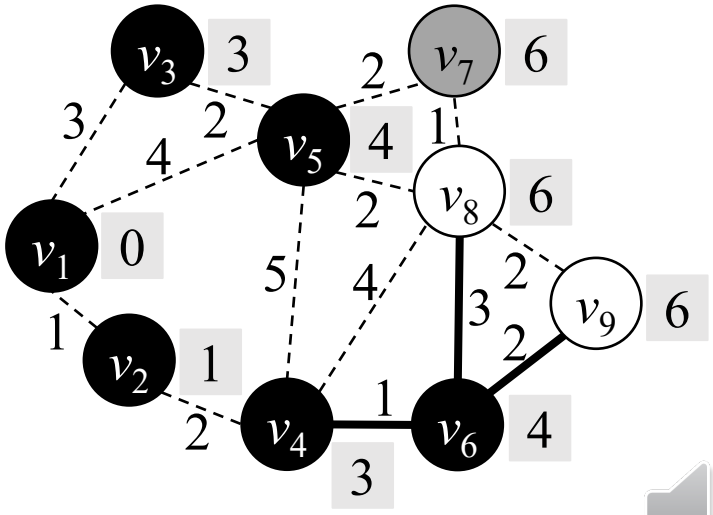
```
1 u.d ← 0;
2 while Q ≠ ∅ do
3     v ← arg min_{w∈Q} w.d;
4     Q ← Q \ {v};
5     foreach (v, w) ∈ E do
6         if w.d > v.d + w((v, w)) then
7             w.d ← v.d + w((v, w));
```

数字：d属性值
灰色顶点：下轮while循环删除的顶点v
黑色顶点：已从Q中删除的顶点
粗实线：顶点v关联的边

戴克斯特拉算法

- 从Q中删除v_6
- 更新v_9的d属性值
- 下轮while循环删除v_7





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

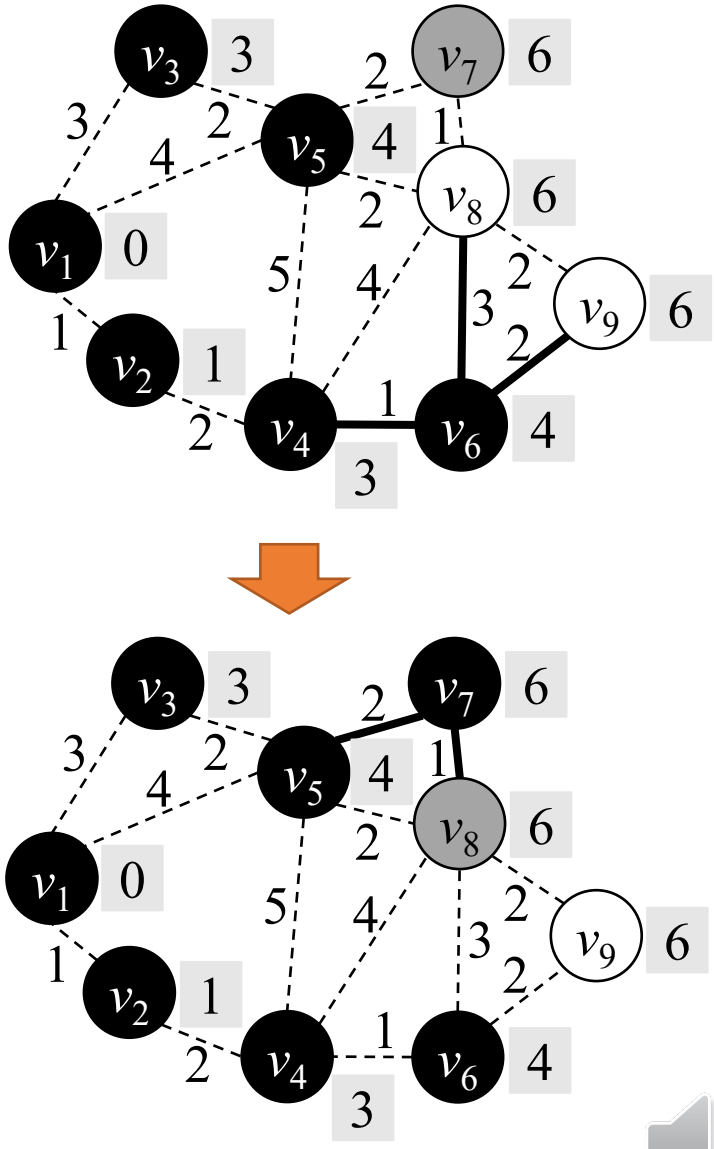
初值: 顶点集 V 中所有顶点的 d 初值为 ∞；集合 Q 初值为 V

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

数字：d属性值
灰色顶点：下轮while循环删除的顶点v
黑色顶点：已从Q中删除的顶点
粗实线：顶点v关联的边

戴克斯特拉算法

- 从Q中删除v_7
- 下轮while循环删除v_8



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

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

戴克斯特拉算法

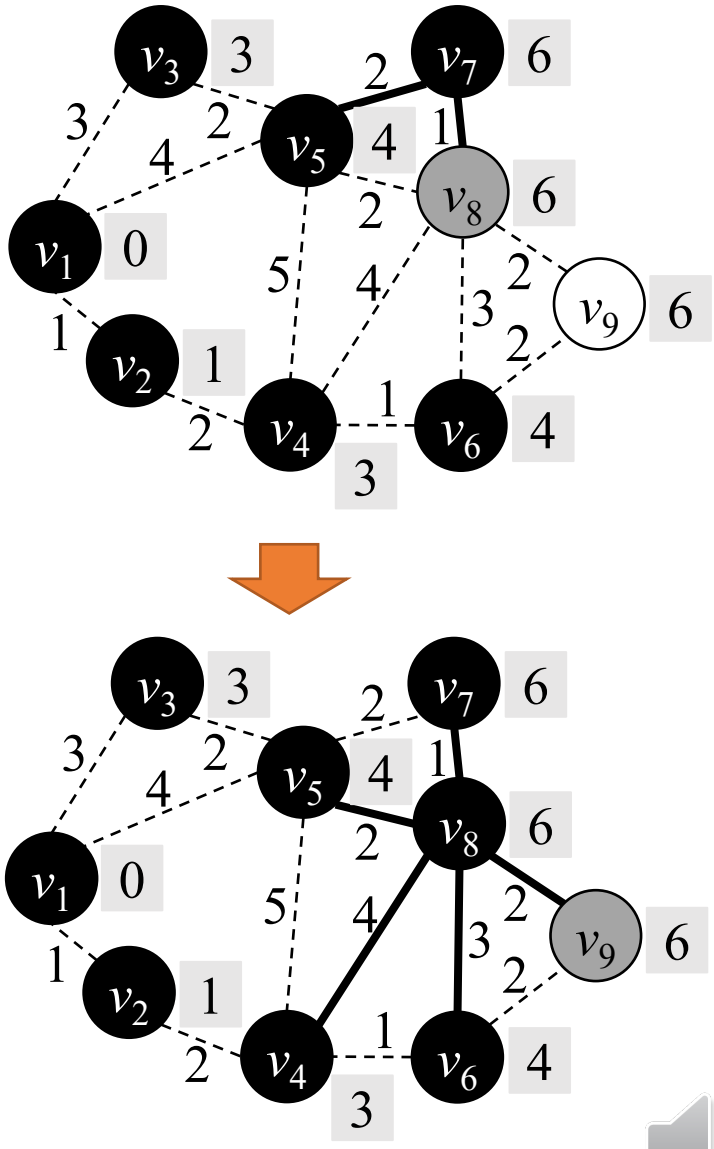
- 从Q中删除v_8
- 下轮while循环删除v_9

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

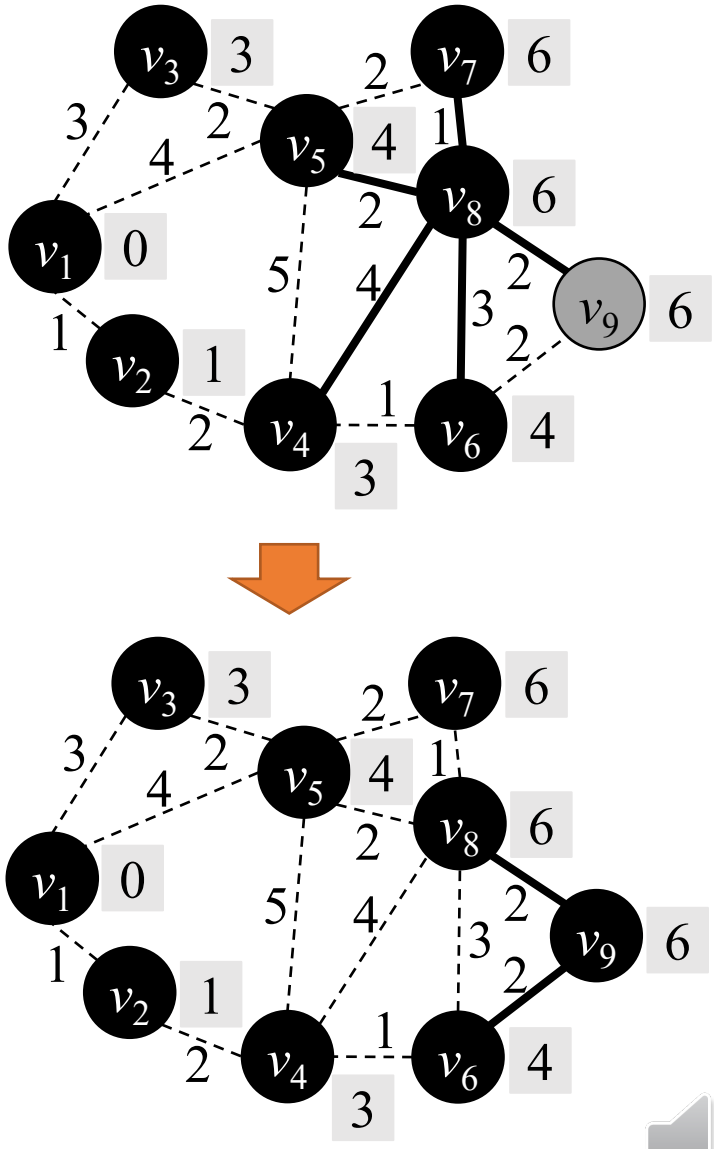
```
1 u.d ← 0;
2 while Q ≠ ∅ do
3     v ← arg min_{w∈Q} w.d;
4     Q ← Q \ {v};
5     foreach (v, w) ∈ E do
6         if w.d > v.d + w((v, w)) then
7             w.d ← v.d + w((v, w));
```



数字：d属性值
灰色顶点：下轮while循环删除的顶点v
黑色顶点：已从Q中删除的顶点
粗实线：顶点v关联的边

戴克斯特拉算法

- 从Q中删除v_9
- 下轮while循环Q为空



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

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

思考题6.2

- 戴克斯特拉算法只适用于边权非负的赋权图。
- 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

```
1 u.d ← 0;
2 while Q ≠ ∅ do
3 |   v ← arg min_{w∈Q} w.d;
4 |   Q ← Q \ {v};
5 |   foreach (v, w) ∈ E do
6 |   |   if w.d > v.d + w((v, w)) then
7 |   |   |   w.d ← v.d + w((v, w));
```

思考题6.2

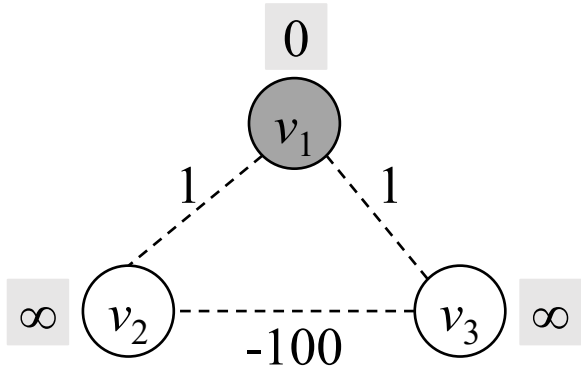
- 戴克斯特拉算法只适用于边权非负的赋权图。
- 对于含负权边的赋权图，戴克斯特拉算法运行结束时，和顶点u连通的顶点的d属性值未必为其和u间的距离，你能举例说明吗？

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

```
1 u.d ← 0;
2 while Q ≠ ∅ do
3 |   v ← arg min_{w∈Q} w.d;
4 |   Q ← Q \ {v};
5 |   foreach (v, w) ∈ E do
6 |   |   if w.d > v.d + w((v, w)) then
7 |   |   |   w.d ← v.d + w((v, w));
```



思考题6.2

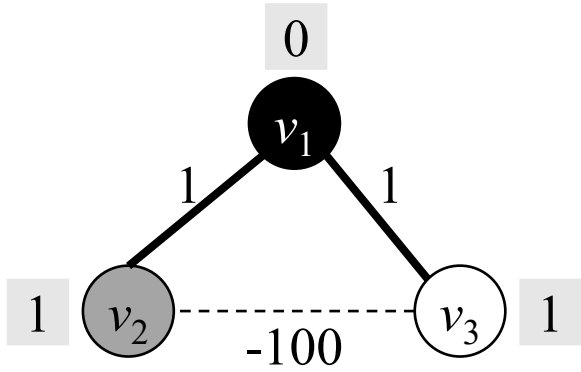
- 戴克斯特拉算法只适用于边权非负的赋权图。
- 对于含负权边的赋权图，戴克斯特拉算法运行结束时，和顶点u连通的顶点的d属性值未必为其和u间的距离，你能举例说明吗？

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

```
1 u.d ← 0;
2 while Q ≠ ∅ do
3 |   v ← arg min_{w∈Q} w.d;
4 |   Q ← Q \ {v};
5 |   foreach (v, w) ∈ E do
6 |   |   if w.d > v.d + w((v, w)) then
7 |   |   |   w.d ← v.d + w((v, w));
```



思考题6.2

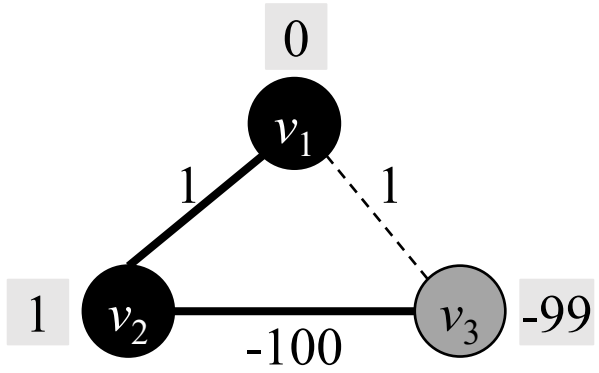
- 戴克斯特拉算法只适用于边权非负的赋权图。
- 对于含负权边的赋权图，戴克斯特拉算法运行结束时，和顶点u连通的顶点的d属性值未必为其和u间的距离，你能举例说明吗？

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

```
1 u.d ← 0;
2 while Q ≠ ∅ do
3 |   v ← arg min_{w∈Q} w.d;
4 |   Q ← Q \ {v};
5 |   foreach (v, w) ∈ E do
6 |   |   if w.d > v.d + w((v, w)) then
7 |   |   |   w.d ← v.d + w((v, w));
```



思考题6.2

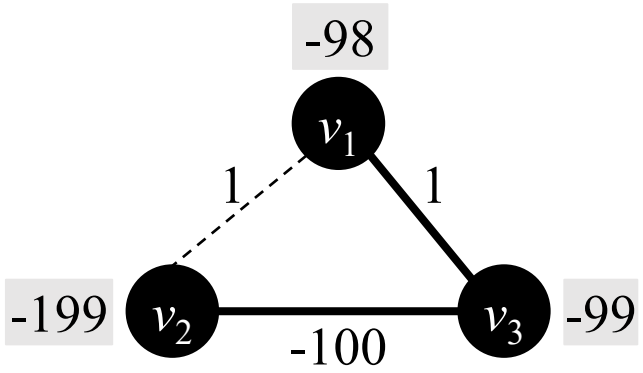
- 戴克斯特拉算法只适用于边权非负的赋权图。
- 对于含负权边的赋权图，戴克斯特拉算法运行结束时，和顶点u连通的顶点的d属性值未必为其和u间的距离，你能举例说明吗？

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

```
1 u.d ← 0;
2 while Q ≠ ∅ do
3 |   v ← arg min_{w∈Q} w.d;
4 |   Q ← Q \ {v};
5 |   foreach (v, w) ∈ E do
6 |   |   if w.d > v.d + w((v, w)) then
7 |   |   |   w.d ← v.d + w((v, w));
```



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$v.\mathrm{d} = \mathrm{dist}(u, v)$。

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

```
1 u.d ← 0;
2 while Q ≠ ∅ do
3 |   v ← arg min_{w∈Q} w.d;
4 |   Q ← Q \ {v};
5 |   foreach (v, w) ∈ E do
6 |   |   if w.d > v.d + w((v, w)) then
7 |   |   |   w.d ← v.d + w((v, w));
```

定理6.1

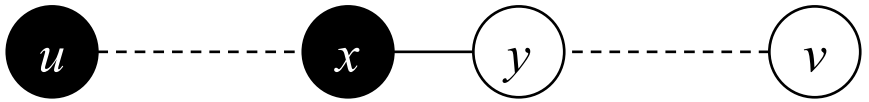
- 戴克斯特拉算法每轮while循环条件判定前，已从集合Q中删除的每个顶点v满足$v.\mathrm{d} = \mathrm{dist}(u, v)$。
  - 采用反证法：假设从集合Q中删除的第一个不满足的顶点为v，则$v \neq u$。

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

```
1 u.d ← 0;
2 while Q ≠ ∅ do
3 |   v ← arg min_{w∈Q} w.d;
4 |   Q ← Q \ {v};
5 |   foreach (v, w) ∈ E do
6 |   |   if w.d > v.d + w((v, w)) then
7 |   |   |   w.d ← v.d + w((v, w));
```



定理6.1

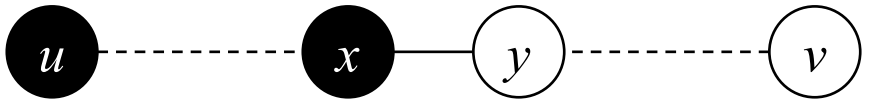
- 戴克斯特拉算法每轮while循环条件判定前，已从集合Q中删除的每个顶点v满足$v.\mathrm{d} = \mathrm{dist}(u, v)$。
  - 采用反证法：假设从集合Q中删除的第一个不满足的顶点为v，则$v \neq u$。
  - 在从Q中删除v的那轮while循环条件判定前，对于一条最短u-v路，
    - 将其经过的第一个未从Q中删除的顶点记作y（可能是v）；
    - 经过的y的前一个邻点记作x（可能是u），该顶点已从Q中删除，因此，$x.\mathrm{d} = \mathrm{dist}(u, x)$。

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

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



定理6.1

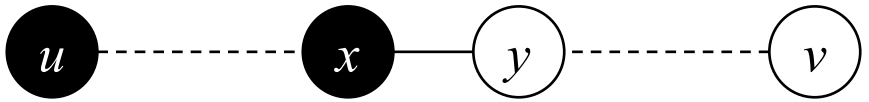
- 戴克斯特拉算法每轮while循环条件判定前，已从集合Q中删除的每个顶点v满足$v.\mathrm{d} = \mathrm{dist}(u, v)$。
  - 采用反证法：假设从集合Q中删除的第一个不满足的顶点为v，则$v \neq u$。
  - 在从Q中删除v的那轮while循环条件判定前，对于一条最短u-v路，
    - 将其经过的第一个未从Q中删除的顶点记作y（可能是v）；
    - 经过的y的前一个邻点记作x（可能是u），该顶点已从Q中删除，因此，$x.\mathrm{d} = \mathrm{dist}(u, x)$。
  - 根据戴克斯特拉算法，$y.\mathrm{d} \leq x.\mathrm{d} + w((x,y)) = \mathrm{dist}(u,y) \leq \mathrm{dist}(u,v) \leq v.\mathrm{d} \leq y.\mathrm{d}$
    则$\mathrm{dist}(u, v) = v.\mathrm{d}$，与$v$不满足矛盾。

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

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



思考题6.4

- 负权边的存在对上述证明过程有何影响？

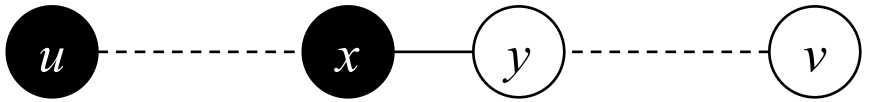
> - 采用反证法：假设从集合Q中删除的第一个不满足的顶点为v，则$v \neq u$。
> - 在从Q中删除v的那轮while循环条件判定前，对于一条最短u-v路，
>   - 将其经过的第一个未从Q中删除的顶点记作y（可能是v）；
>   - 经过的y的前一个邻点记作x（可能是u），该顶点已从Q中删除，因此，$x.\mathrm{d} = \mathrm{dist}(u, x)$。
> - 根据戴克斯特拉算法，$y.\mathrm{d} \leq x.\mathrm{d} + w\big((x,y)\big) = \mathrm{dist}(u,y) \leq \mathrm{dist}(u,v) \leq v.\mathrm{d} \leq y.\mathrm{d}$
>
> 则$\mathrm{dist}(u, v) = v.\mathrm{d}$，与$v$不满足矛盾。

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

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



思考题6.4

- 负权边的存在对上述证明过程有何影响？

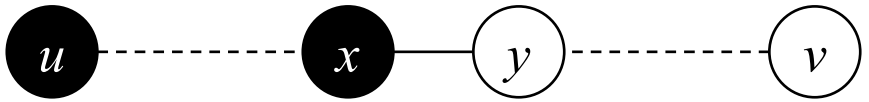
  - 采用反证法：假设从集合Q中删除的第一个不满足的顶点为v，则$v \neq u$。
  - 在从Q中删除v的那轮while循环条件判定前，对于一条最短u-v路，
    - 将其经过的第一个未从Q中删除的顶点记作y（可能是v）；
    - 经过的y的前一个邻点记作x（可能是u），该顶点已从Q中删除，因此，$x.\mathrm{d} = \mathrm{dist}(u, x)$。
  - 根据戴克斯特拉算法，$y.\mathrm{d} \leq x.\mathrm{d} + w\big((x,y)\big) = \mathrm{dist}(u,y) \leq \mathrm{dist}(u,v) \leq v.\mathrm{d} \leq y.\mathrm{d}$
    则$\mathrm{dist}(u, v) = v.\mathrm{d}$，与$v$不满足矛盾。

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

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



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

```
1 u.d ← 0;
2 while Q ≠ ∅ do
3 |   v ← arg min_{w∈Q} w.d;
4 |   Q ← Q \ {v};
5 |   foreach (v, w) ∈ E do
6 |   |   if w.d > v.d + w((v, w)) then
7 |   |   |   w.d ← v.d + w((v, w));
```

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？
  - 每轮while循环从Q中选择d属性值最小的顶点v
  - 剩余顶点当前和未来的d属性值不会小于v.d

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

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

戴克斯特拉算法

- 时间复杂度：$O((n+m)\log n)$
  - 构造Q（基于二叉堆的优先队列）：$O(n)$
  - 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d属性值并维护Q：$O(m\log n)$

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

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

贝尔曼-福特算法 和 弗洛伊德-沃舍尔算法

- 对于含负权边的赋权图，可以通过贝尔曼-福特算法计算距离。
- 戴克斯特拉算法可以计算赋权图中一个顶点和所有顶点间的距离。
  若要计算赋权图中所有顶点间的距离，则可以通过时间复杂度更低的弗洛伊德-沃舍尔算法。

请认真完成课后练习