

图算法篇：最小生成树I

童咏昕

北京航空航天大学
计算机学院

中国大学MOOC北航《算法设计与分析》

问题背景

通用框架

Prim算法

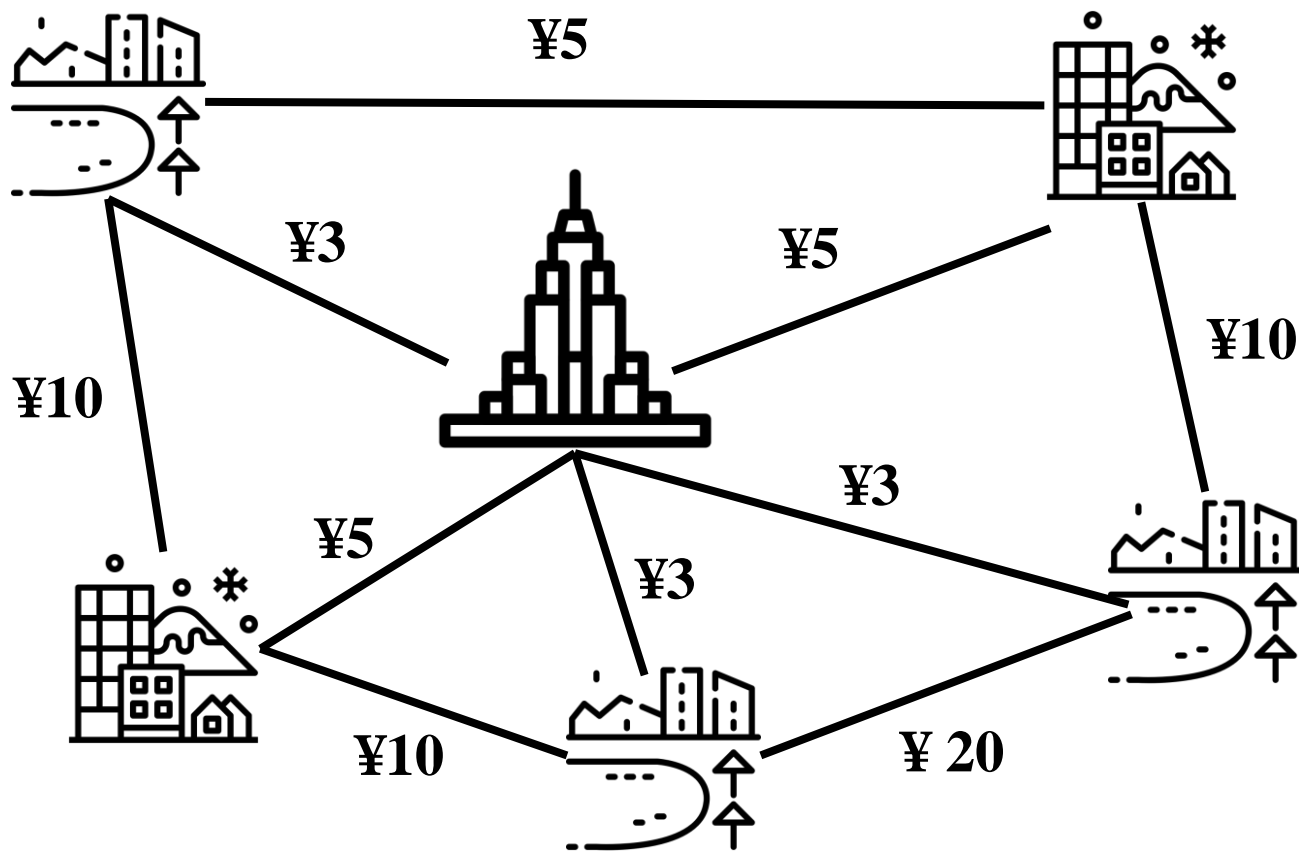
算法实例

算法分析

问题背景：道路修建



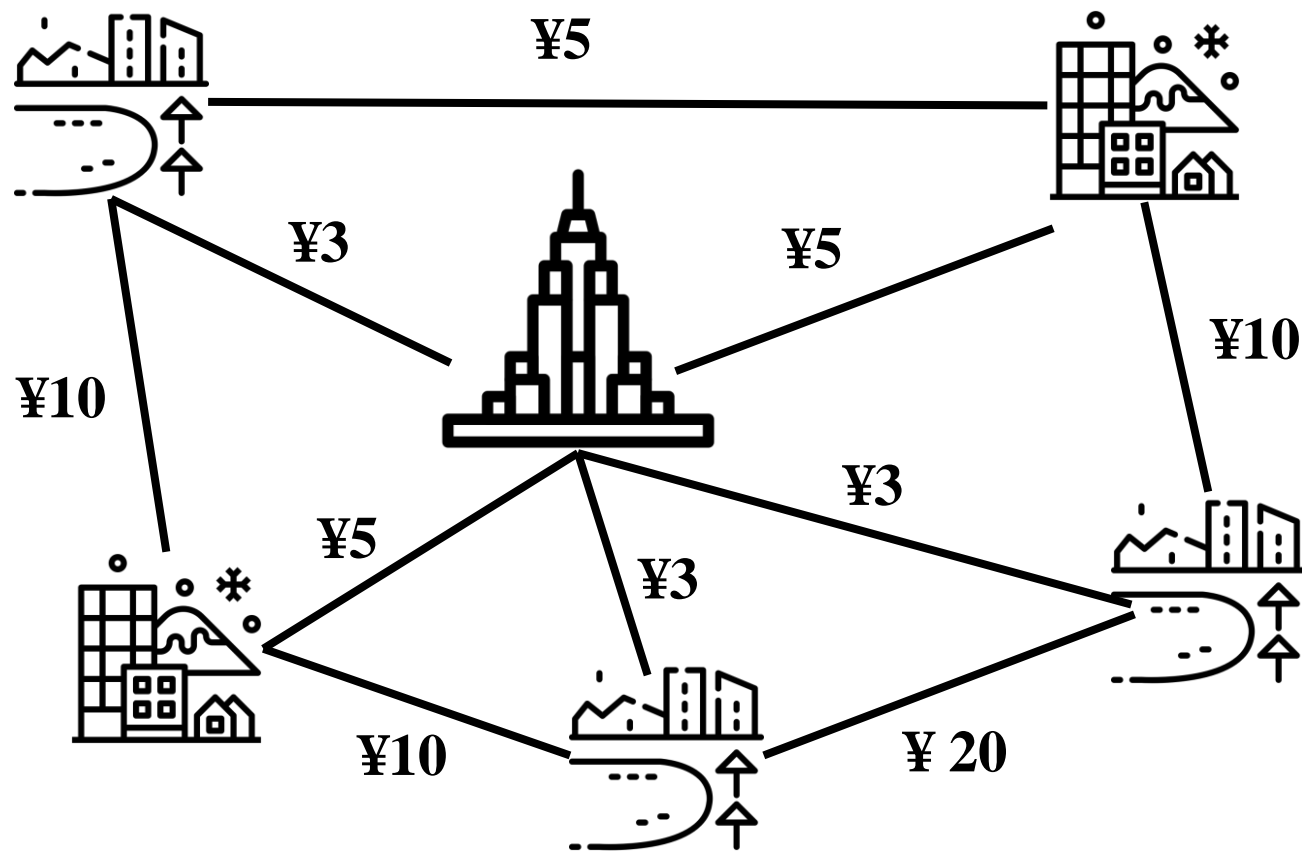
- 需要修建道路连通城市，各道路花费不同



问题背景：道路修建



- 需要修建道路连通城市，各道路花费不同



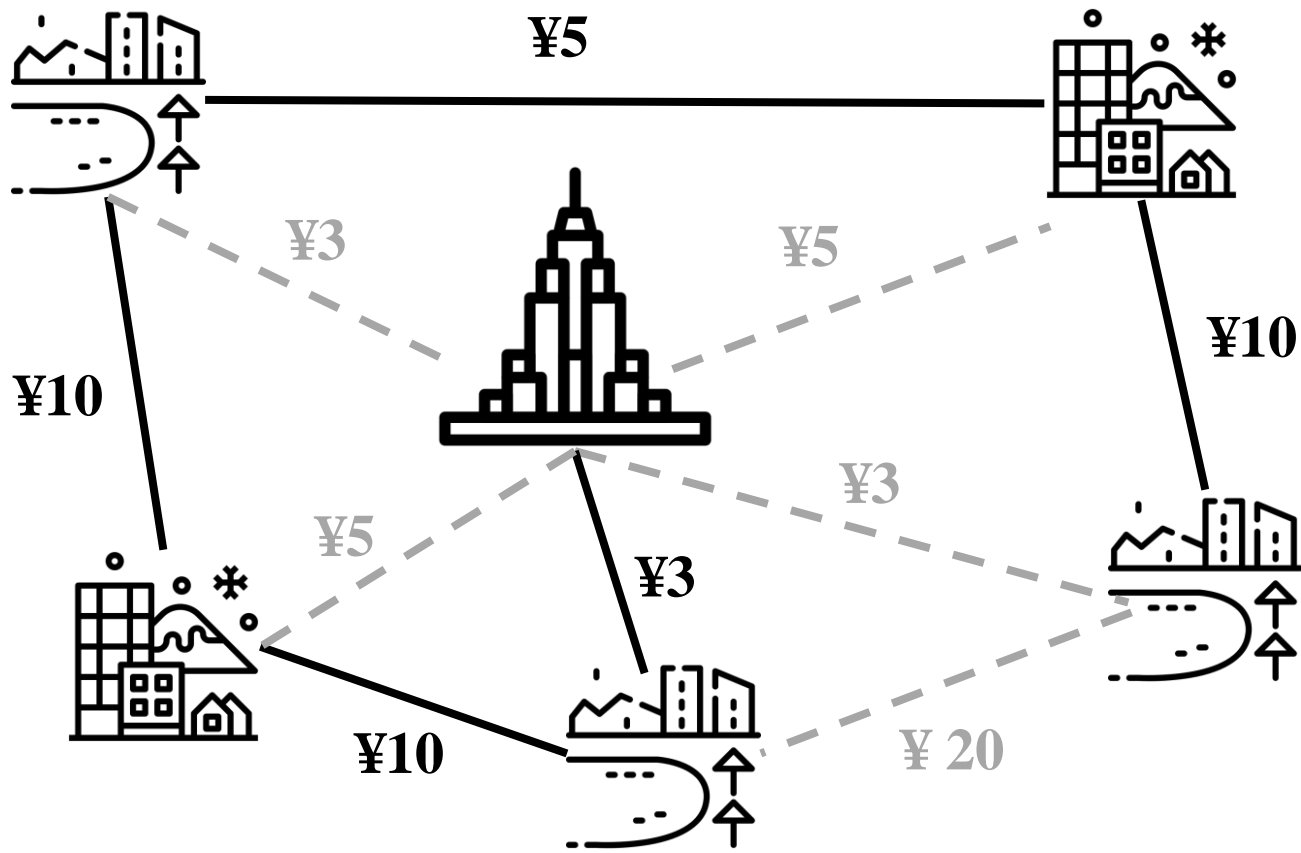
方案	花费
	¥74

$$\text{花费: } 10 + 10 + 10 + 5 + 20 + 3 + 3 + 5 + 3 + 5 = 74$$

问题背景：道路修建



- 需要修建道路连通城市，各道路花费不同



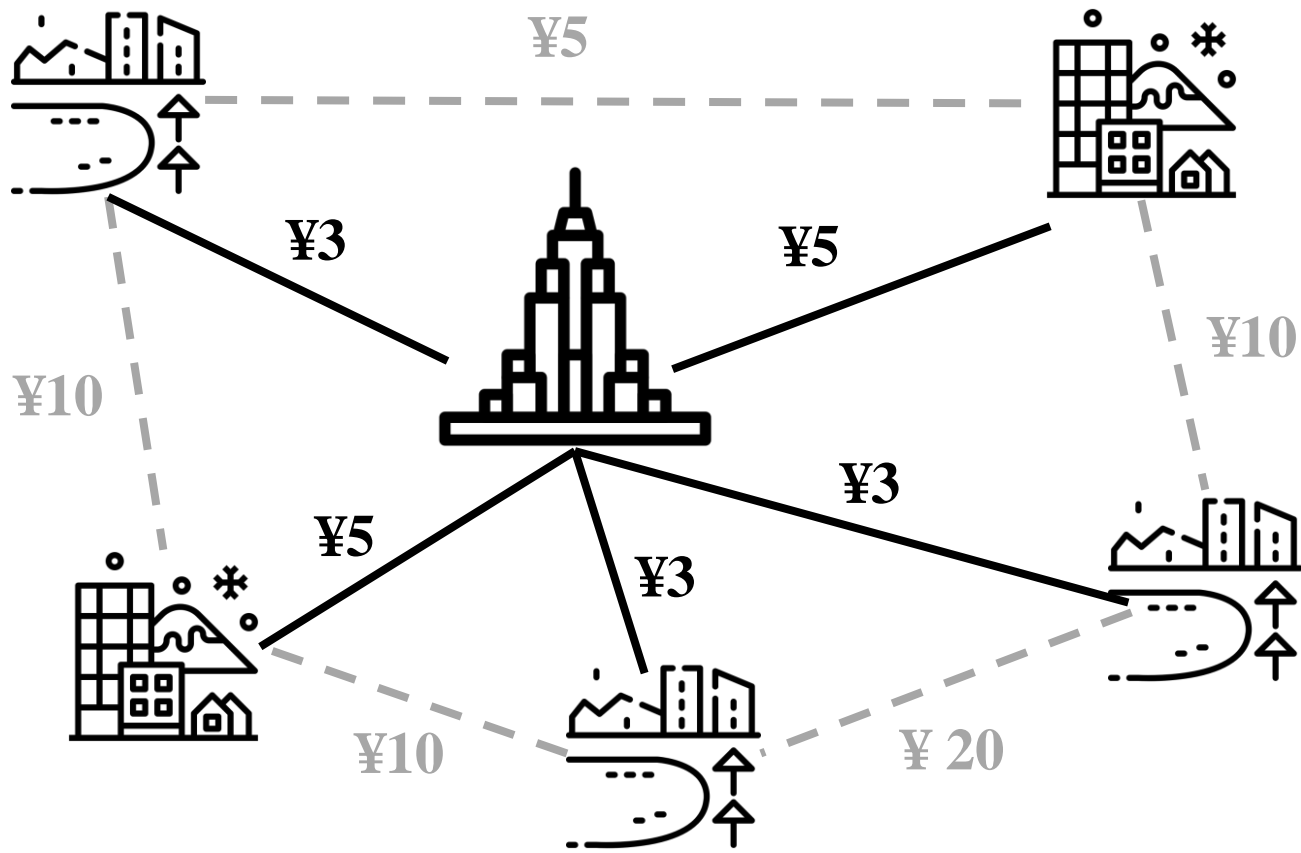
花费：10 + 10 + 10 + 5 + 3 = 38

方案	花费
	¥74
	¥38

问题背景：道路修建



- 需要修建道路连通城市，各道路花费不同

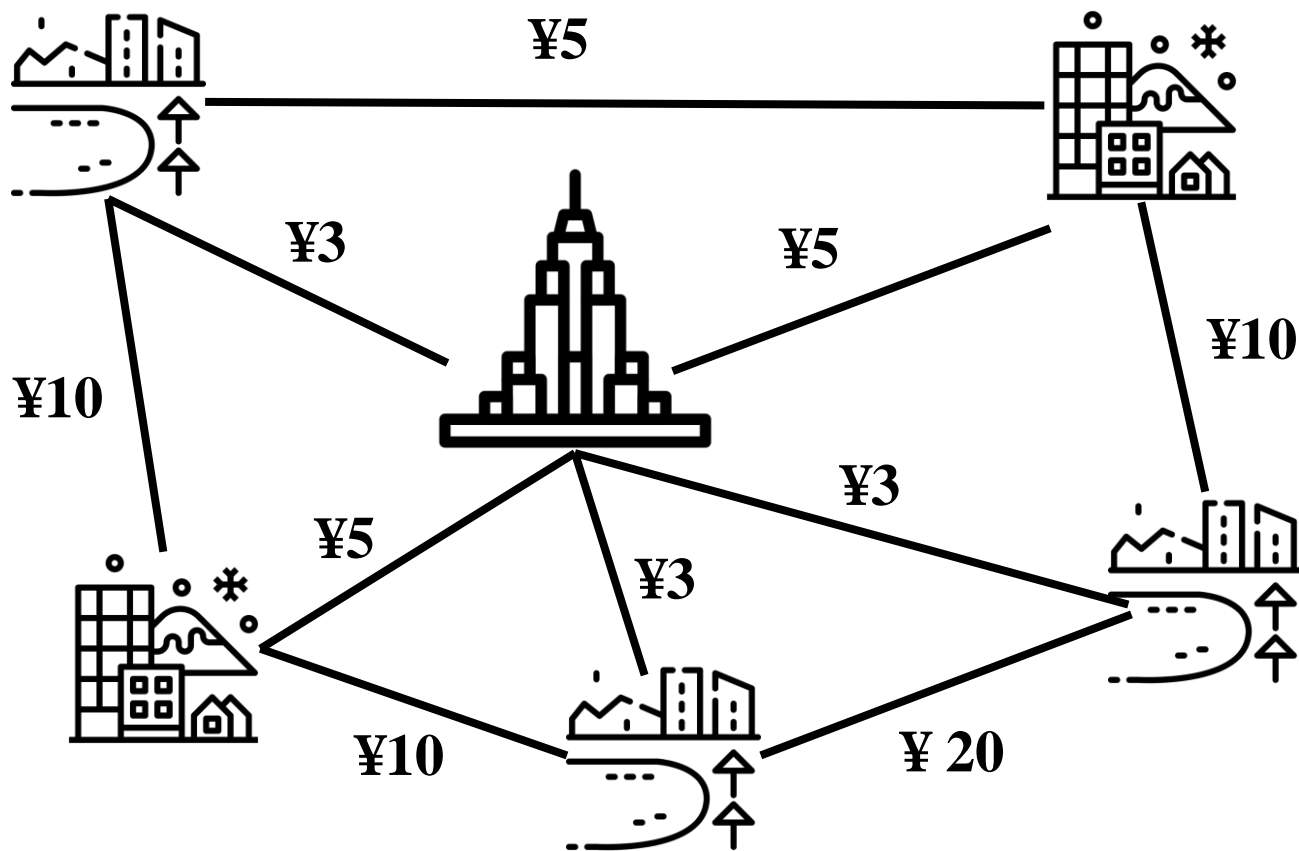


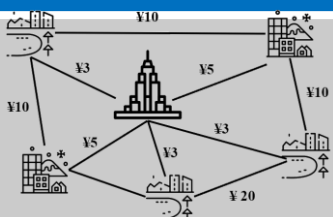
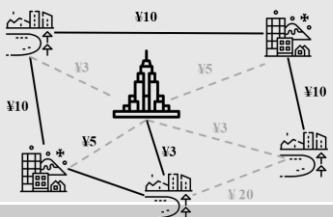
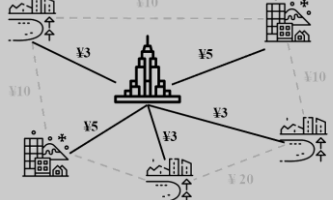
方案	花费
	¥74
	¥38
	¥19

花费： $3 + 3 + 5 + 3 + 5 = 19$

问题背景：道路修建

- 需要修建道路连通城市，各道路花费不同

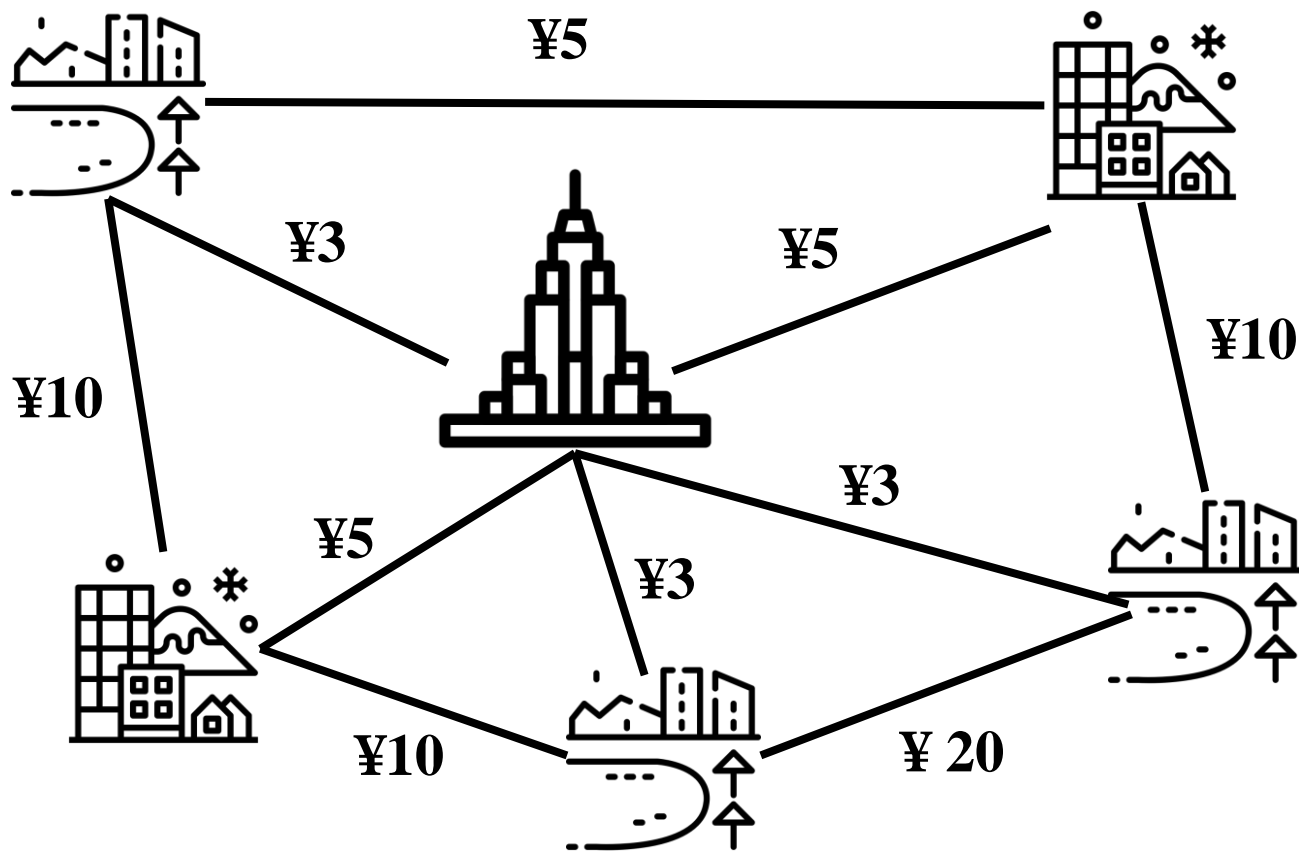


方案	花费
	¥74
	¥38
	¥19

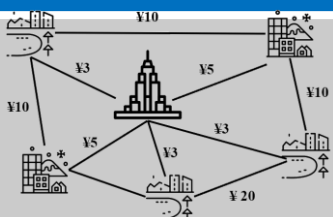
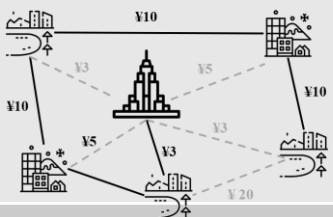
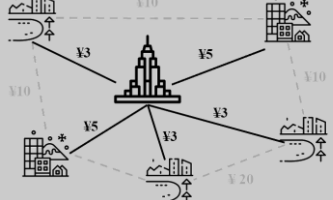
问题：连通各城市的最小花费是多少？

问题背景：道路修建

- 需要修建道路连通城市，各道路花费不同



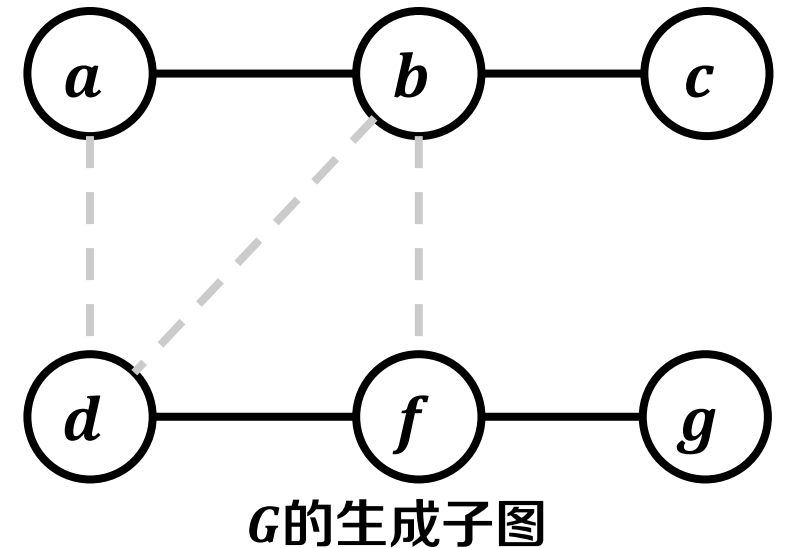
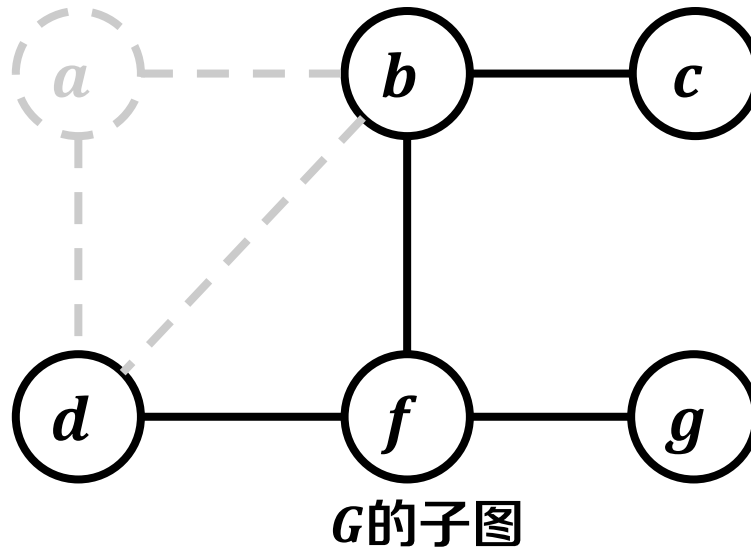
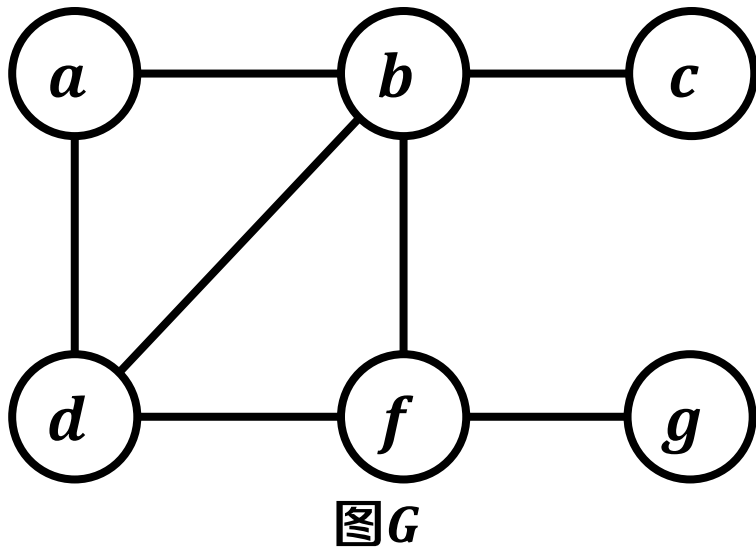
问题：连通各城市的最小花费是多少？

方案	花费
	¥74
	¥38
	¥19

权重最小的连通生成子图

图的概念回顾：生成子图

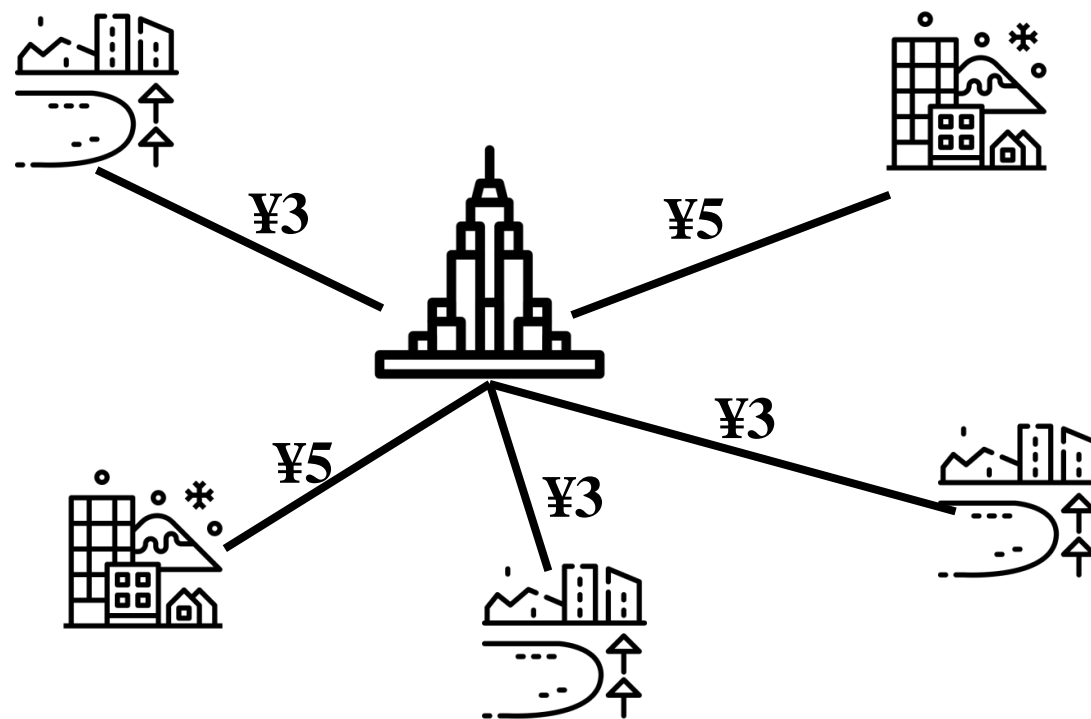
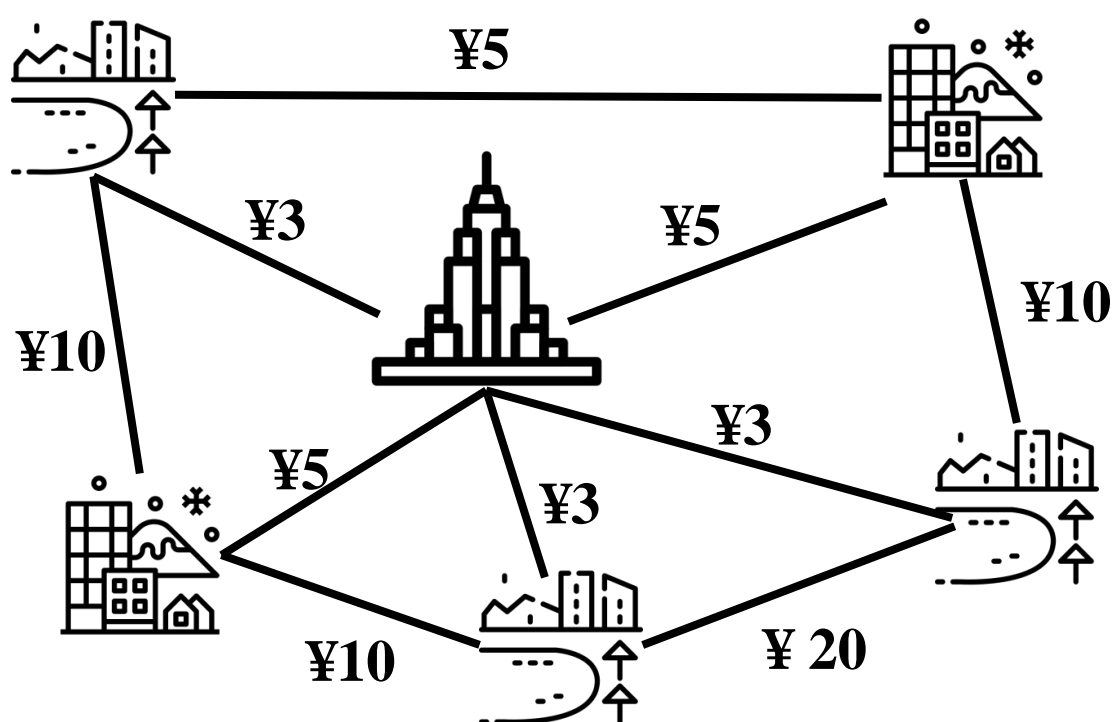
- 子图(Subgraph)
 - 如果 $V' \subseteq V, E' \subseteq E$ ，则称图 $G' = \langle V', E' \rangle$ 是图 G 的一个子图
- 生成子图(Spanning Subgraph)
 - 如果 $V' = V, E' \subseteq E$ ，则称图 $G' = \langle V', E' \rangle$ 是图 G 的一个生成子图



图的概念：生成树

- 生成树(Spanning Tree)

- 图 $T' = \langle V', E' \rangle$ 是无向图 G 的一个生成子图，并且是连通、无环路的(树)



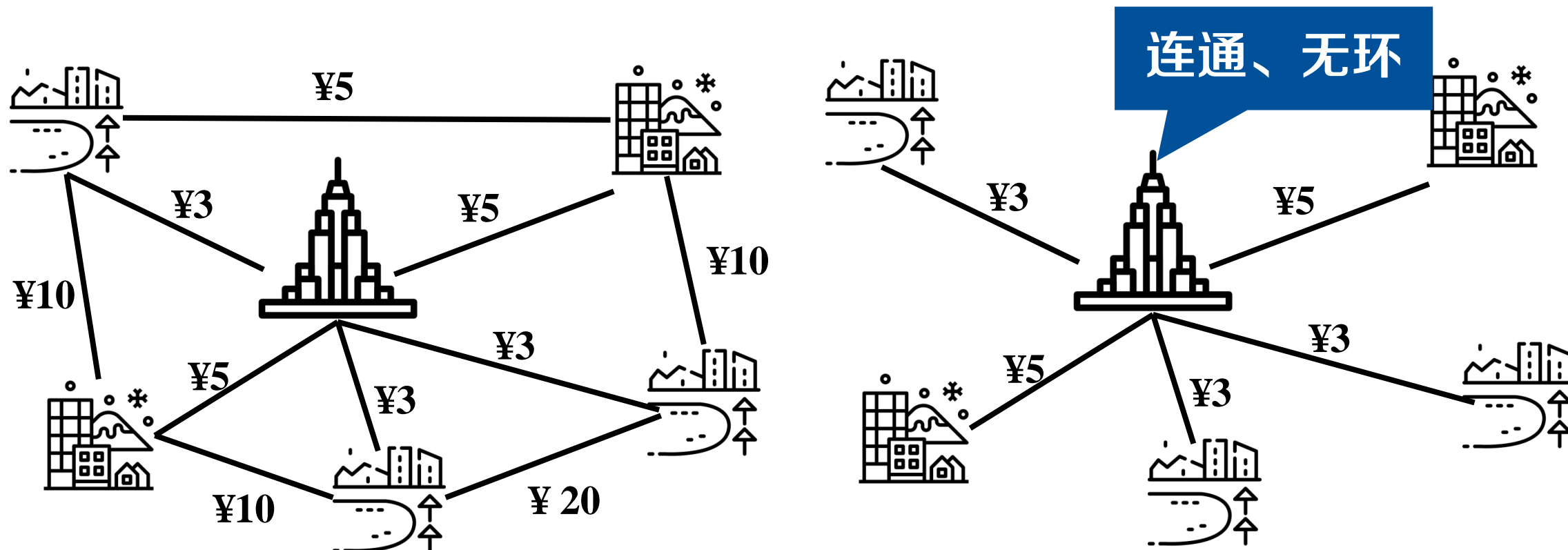
问题：连通各城市的最小花费是多少？

权重最小的连通生成子图

图的概念：生成树

- 生成树(Spanning Tree)

- 图 $T' = \langle V', E' \rangle$ 是无向图 G 的一个生成子图，并且是连通、无环路的(树)



问题：连通各城市的最小花费是多少？

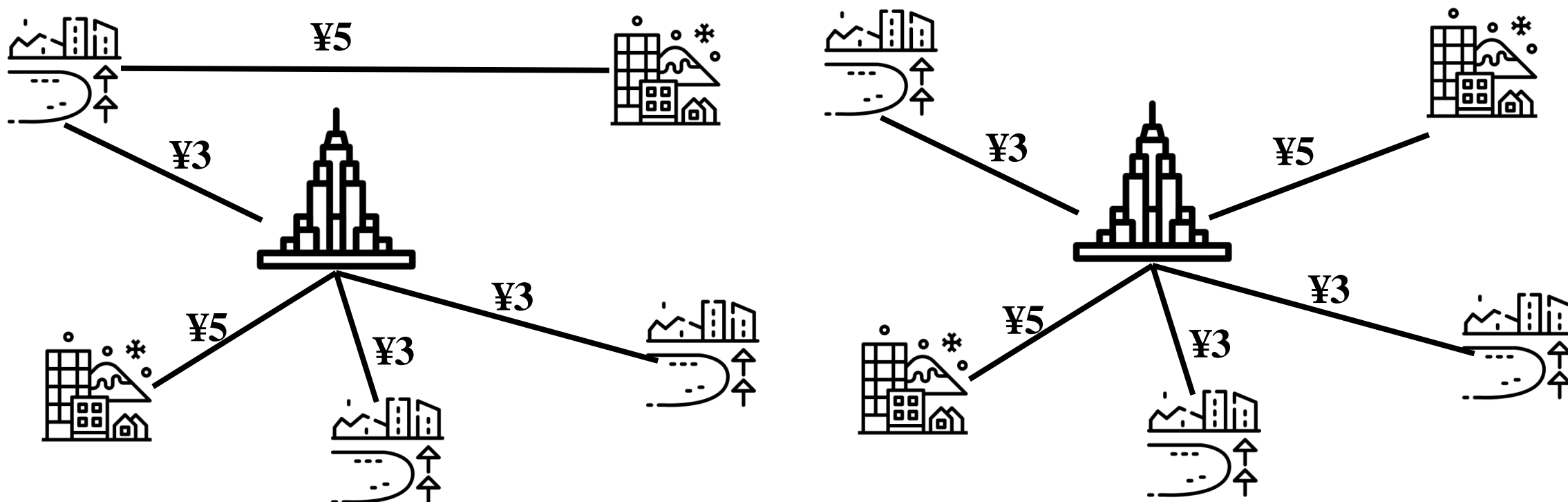
权重最小的生成树

图的概念：生成树



- 生成树(Spanning Tree)

- 图 $T' = \langle V', E' \rangle$ 是无向图 G 的一个生成子图，并且是连通、无环路的(树)



权重最小的生成树可能**不唯一**！

最小生成树问题

Minimum Spanning Tree Problem

输入

- 连通无向图 $G = \langle V, E, W \rangle$, 其中 $w(u, v) \in W$ 表示边 (u, v) 的权重

最小生成树问题

Minimum Spanning Tree Problem

输入

- 连通无向图 $G = \langle V, E, W \rangle$, 其中 $w(u, v) \in W$ 表示边 (u, v) 的权重

输出

- 图 G 的最小生成树 $T = \langle V_T, E_T \rangle$

最小生成树问题

Minimum Spanning Tree Problem

输入

- 连通无向图 $G = \langle V, E, W \rangle$, 其中 $w(u, v) \in W$ 表示边 (u, v) 的权重

输出

- 图 G 的最小生成树 $T = \langle V_T, E_T \rangle$

$$\min \sum_{e \in E_T} w(e)$$

$$s. t. \quad V_T = V, E_T \subseteq E$$

最小生成树问题

Minimum Spanning Tree Problem

输入

- 连通无向图 $G = \langle V, E, W \rangle$, 其中 $w(u, v) \in W$ 表示边 (u, v) 的权重

输出

- 图 G 的最小生成树 $T = \langle V_T, E_T \rangle$

$$\min \sum_{e \in E_T} w(e)$$

优化目标

$$s. t. \quad V_T = V, E_T \subseteq E$$

最小生成树问题

Minimum Spanning Tree Problem

输入

- 连通无向图 $G = \langle V, E, W \rangle$, 其中 $w(u, v) \in W$ 表示边 (u, v) 的权重

输出

- 图 G 的最小生成树 $T = \langle V_T, E_T \rangle$

$$\min \sum_{e \in E_T} w(e)$$

优化目标

$$s. t. \quad V_T = V, E_T \subseteq E$$

约束条件

问题背景

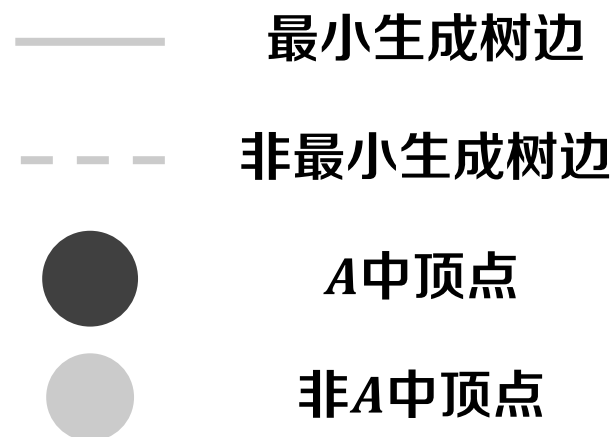
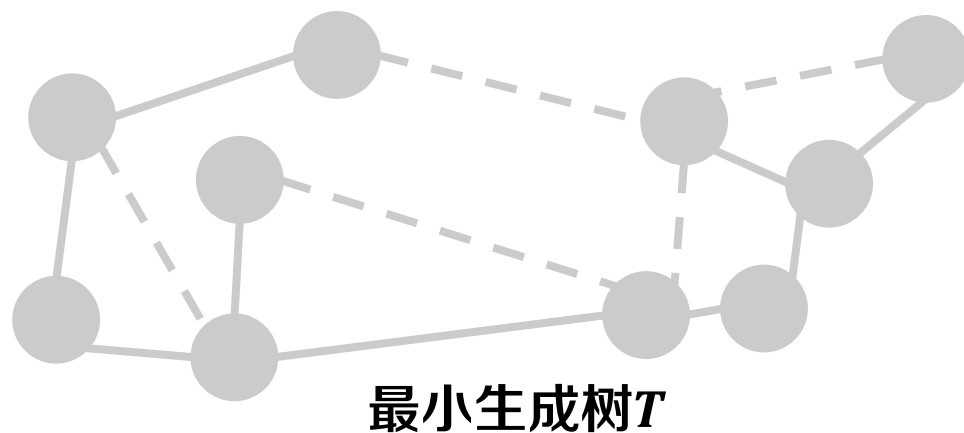
通用框架

Prim算法

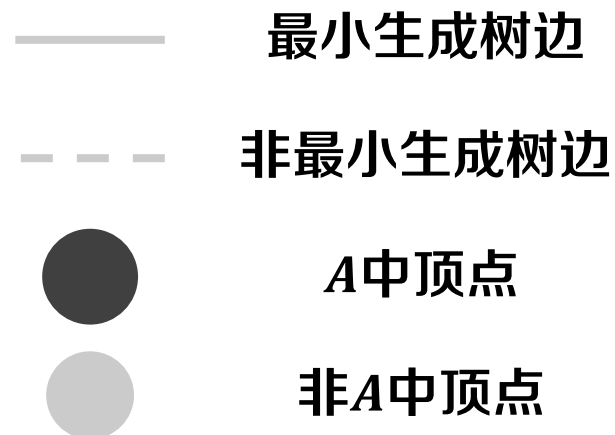
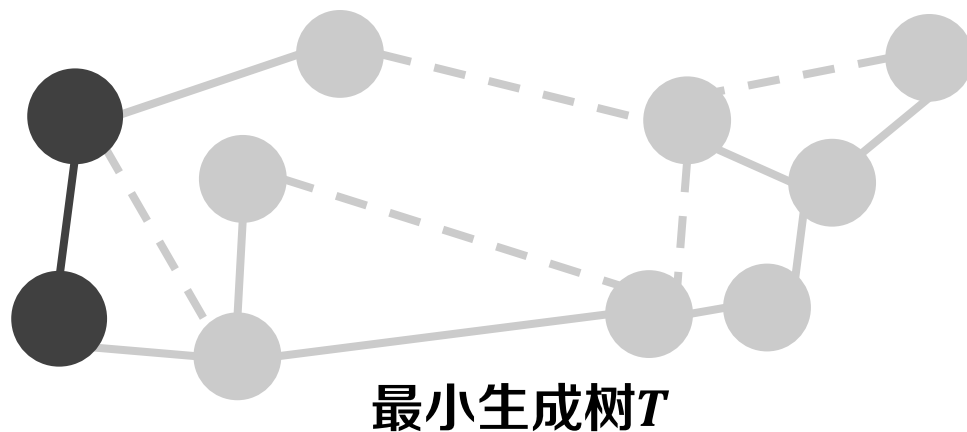
算法实例

算法分析

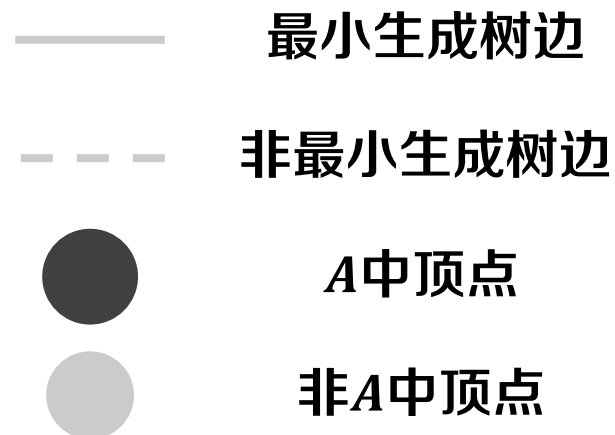
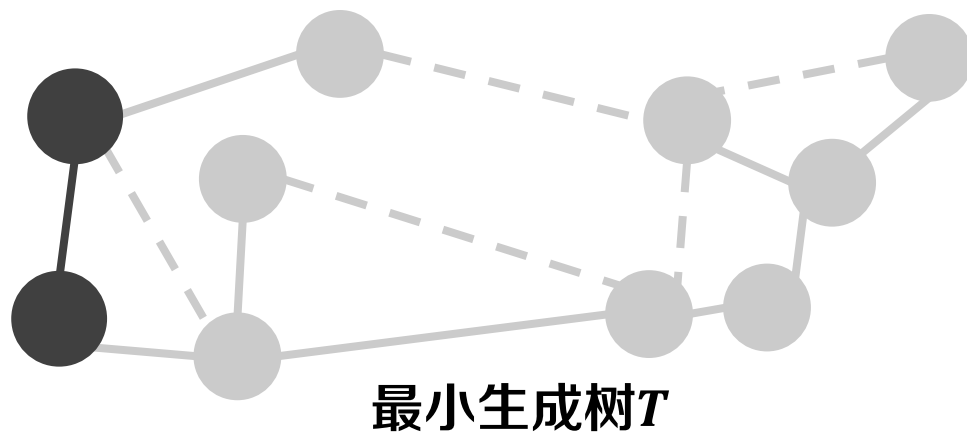
- 生成树是一个无向图中的连通、**无环**的生成子图
 - 新建一个空边集 A ，边集 A 可逐步扩展为最小生成树



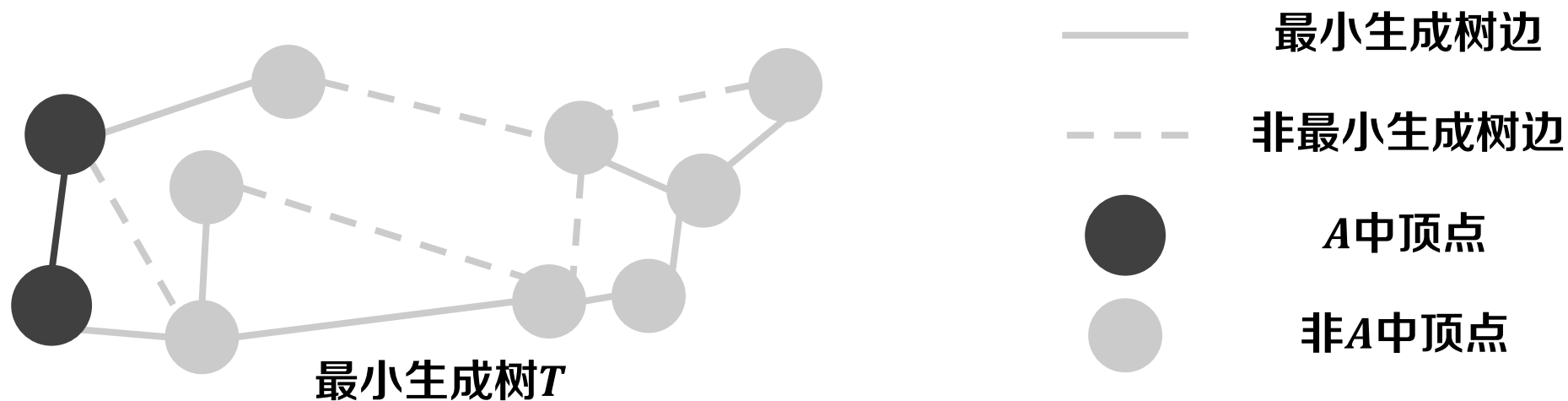
- 生成树是一个无向图中的连通、**无环**的生成子图
 - 新建一个空边集 A ，边集 A 可逐步扩展为最小生成树
 - 每次向边集 A 中新增加一条边



- 生成树是一个无向图中的连通、**无环**的生成子图
 - 新建一个空边集 A ，边集 A 可逐步扩展为最小生成树
 - 每次向边集 A 中新增加一条边
 - 需保证边集 A 仍是一个**无环图**
 - 需保证边集 A 仍是**最小生成树的子集**



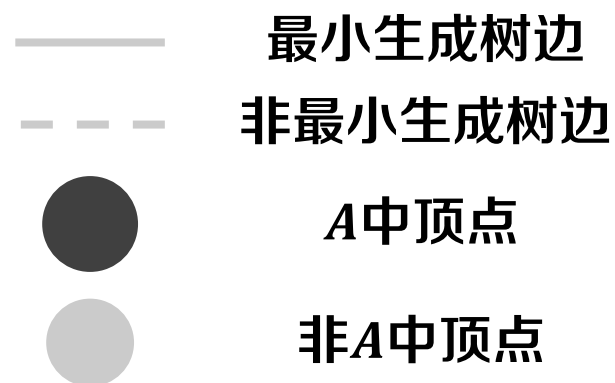
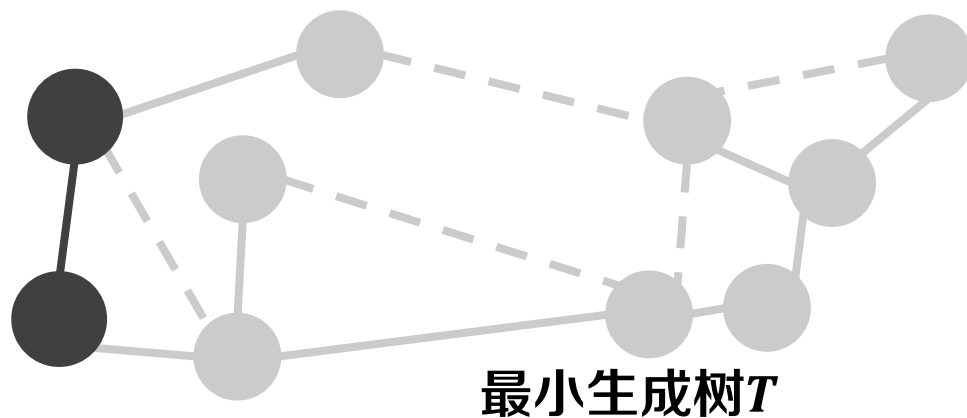
- 生成树是一个无向图中的连通、**无环**的生成子图
 - 新建一个空边集 A ，边集 A 可逐步扩展为最小生成树
 - 每次向边集 A 中新增加一条边
 - 需保证边集 A 仍是一个**无环图**
 - 需保证边集 A 仍是最小生成树的子集



问题：如何保证边集 A 仍是最小生成树的子集？

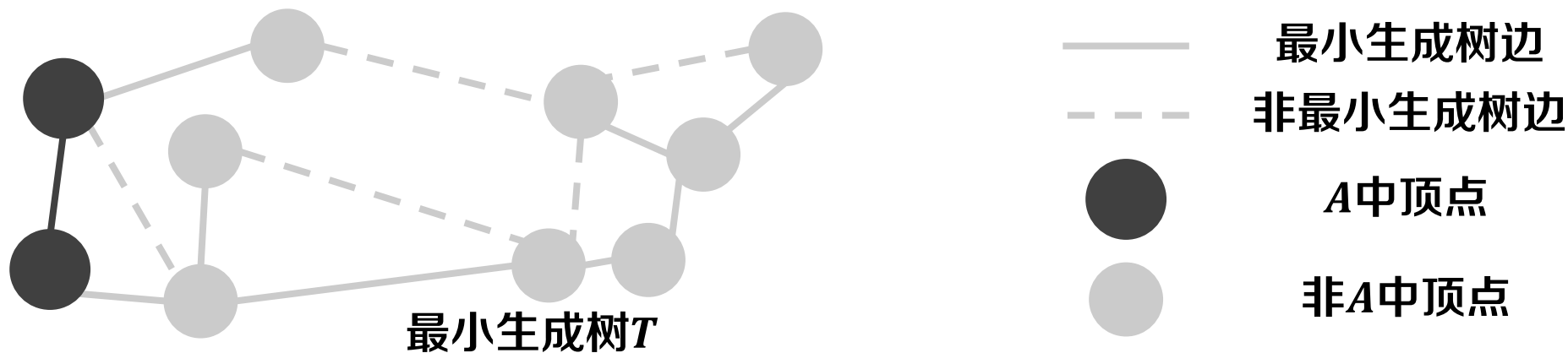
- 安全边(Safe Edge)

- A 是某棵最小生成树 T 边的子集, $A \subseteq T$
- $A \cup \{(u, v)\}$ 仍是 T 边的一个子集, 则称 (u, v) 是 A 的**安全边**



- 安全边(Safe Edge)

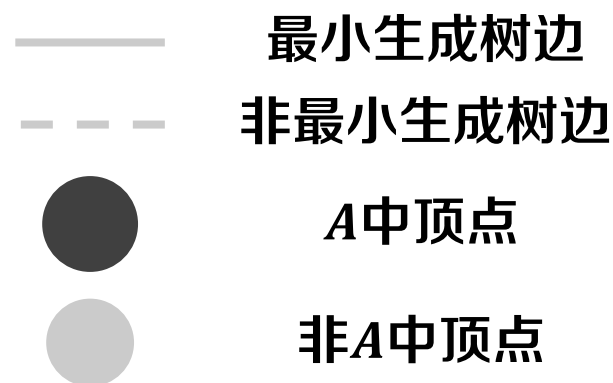
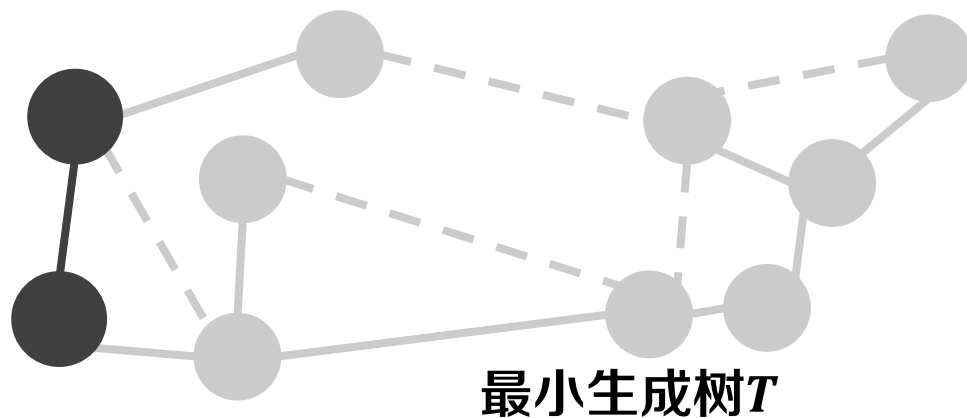
- A 是某棵最小生成树 T 边的子集, $A \subseteq T$
- $A \cup \{(u, v)\}$ 仍是 T 边的一个子集, 则称 (u, v) 是 A 的**安全边**



若每次向边集 A 中新增**安全边**, 可保证边集 A 是最小生成树的子集

- 安全边(Safe Edge)

- A 是某棵最小生成树 T 边的子集, $A \subseteq T$
- $A \cup \{(u, v)\}$ 仍是 T 边的一个子集, 则称 (u, v) 是 A 的**安全边**

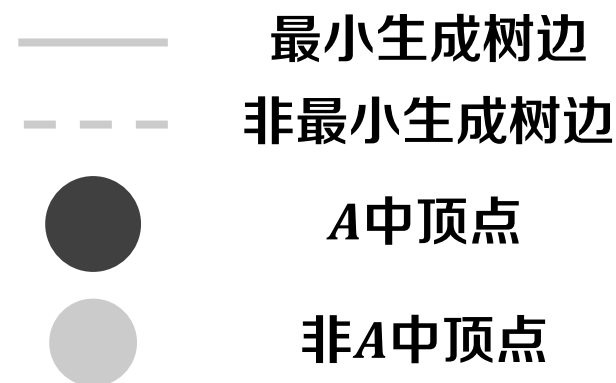
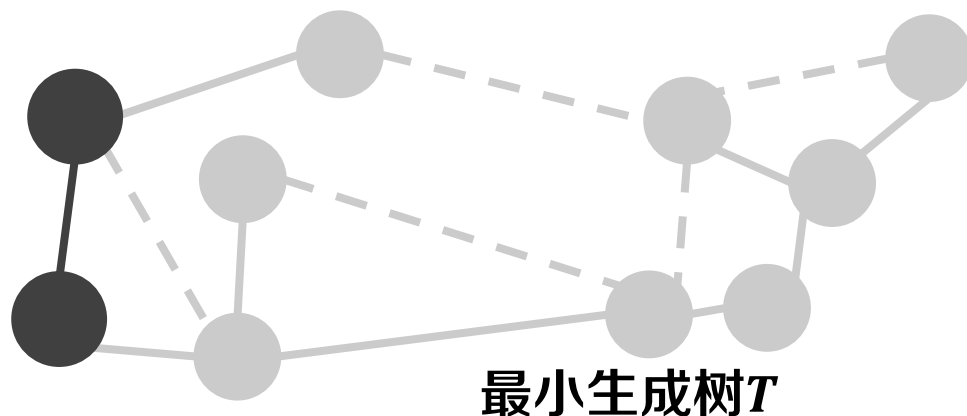


- Generic-MST(G)

```
A ← ∅  
while 没有形成最小生成树 do  
    | 寻找 $A$ 的安全边 $(u, v)$   
    |  $A \leftarrow A \cup (u, v)$   
end  
return  $A$ 
```

- 安全边(Safe Edge)

- A 是某棵最小生成树 T 边的子集, $A \subseteq T$
- $A \cup \{(u, v)\}$ 仍是 T 边的一个子集, 则称 (u, v) 是 A 的**安全边**



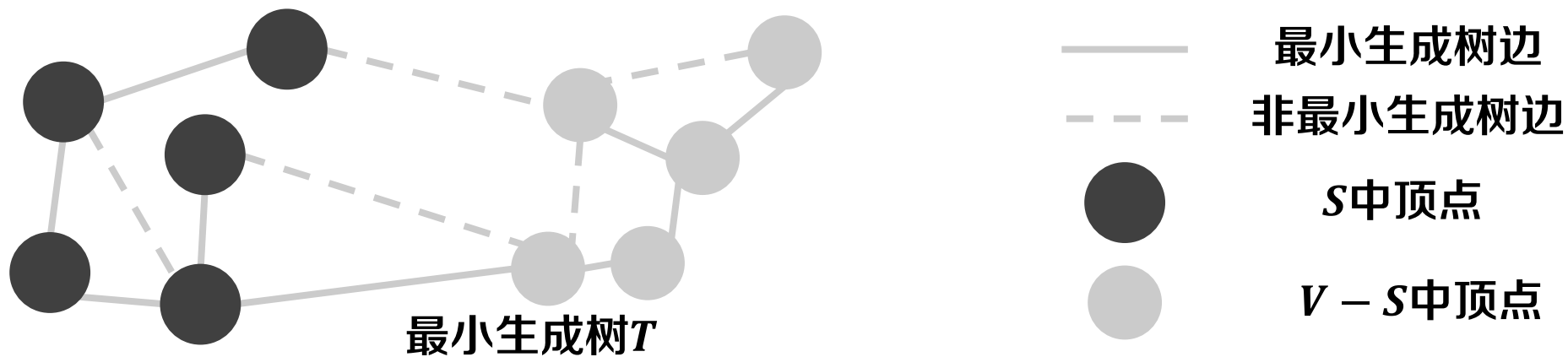
- Generic-MST(G)

```
A ← ∅  
while 没有形成最小生成树 do  
    | 寻找 $A$ 的安全边 $(u, v)$   
    |  $A \leftarrow A \cup (u, v)$   
end  
return  $A$ 
```

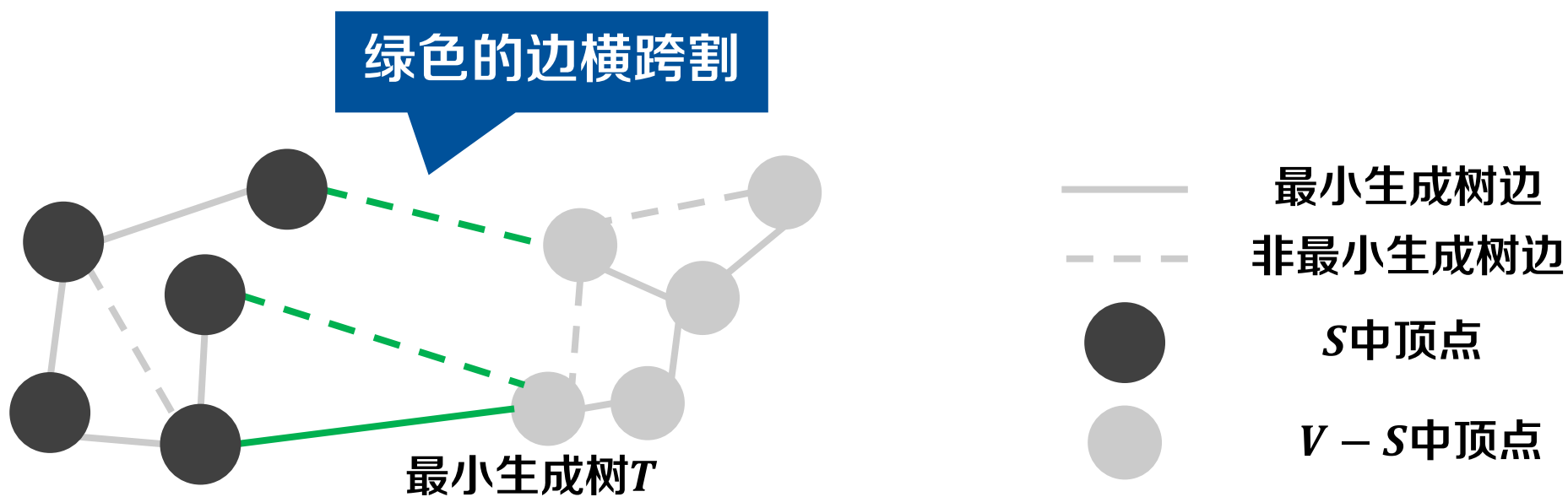
问题：如何有效辨识安全边？

- 割(Cut)

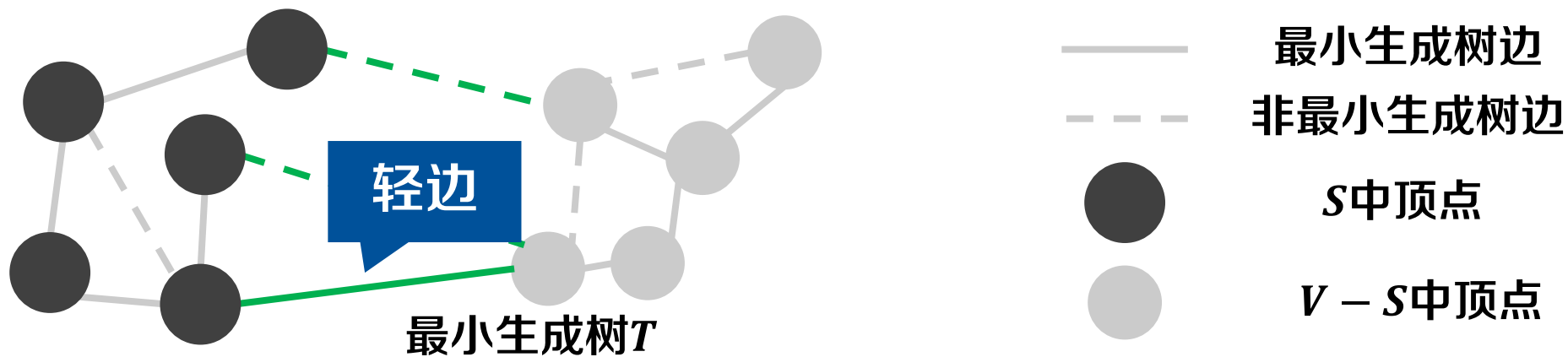
- 图 $G = \langle V, E \rangle$ 是一个连通无向图，**割** $(S, V - S)$ 将图 G 的顶点集 V 划分为两部分



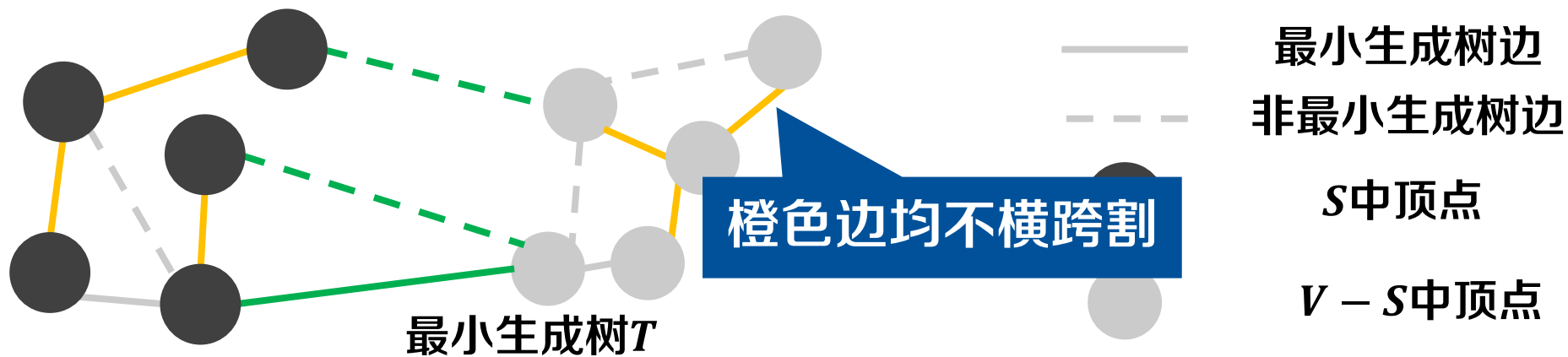
- 割(Cut)
 - 图 $G = \langle V, E \rangle$ 是一个连通无向图，割 $(S, V - S)$ 将图 G 的顶点集 V 划分为两部分
- 横跨(Cross)
 - 给定割 $(S, V - S)$ 和边 (u, v) ， $u \in S$ ， $v \in V - S$ ，称边 (u, v) **横跨** 割 $(S, V - S)$



- 割(Cut)
 - 图 $G = \langle V, E \rangle$ 是一个连通无向图，割 $(S, V - S)$ 将图 G 的顶点集 V 划分为两部分
- 横跨(Cross)
 - 给定割 $(S, V - S)$ 和边 (u, v) ， $u \in S$ ， $v \in V - S$ ，称边 (u, v) 横跨割 $(S, V - S)$
- 轻边(Light Edge)
 - 横跨割的所有边中，权重最小的称为横跨这个割的一条**轻边**



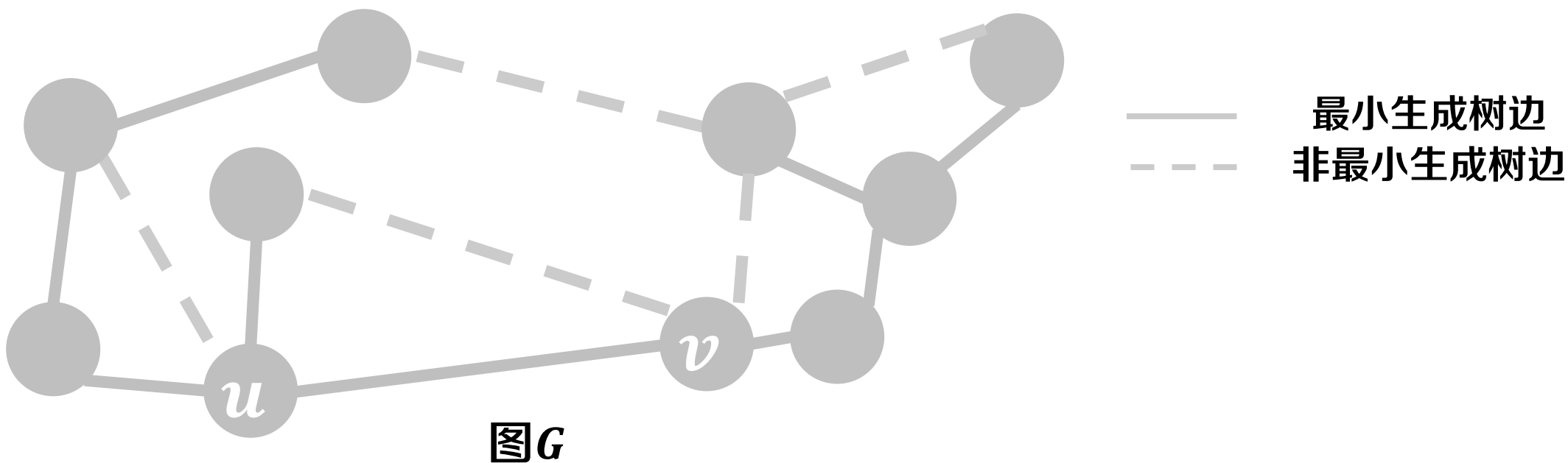
- 割(Cut)
 - 图 $G = \langle V, E \rangle$ 是一个连通无向图，割 $(S, V - S)$ 将图 G 的顶点集 V 划分为两部分
- 横跨(Cross)
 - 给定割 $(S, V - S)$ 和边 (u, v) ， $u \in S$ ， $v \in V - S$ ，称边 (u, v) 横跨割 $(S, V - S)$
- 轻边(Light Edge)
 - 横跨割的所有边中，权重最小的称为横跨这个割的一条轻边
- 不妨害(Respect)
 - 如果一个边集 A 中没有边横跨某割，则称该割**不妨害**边集 A



安全边辨识定理



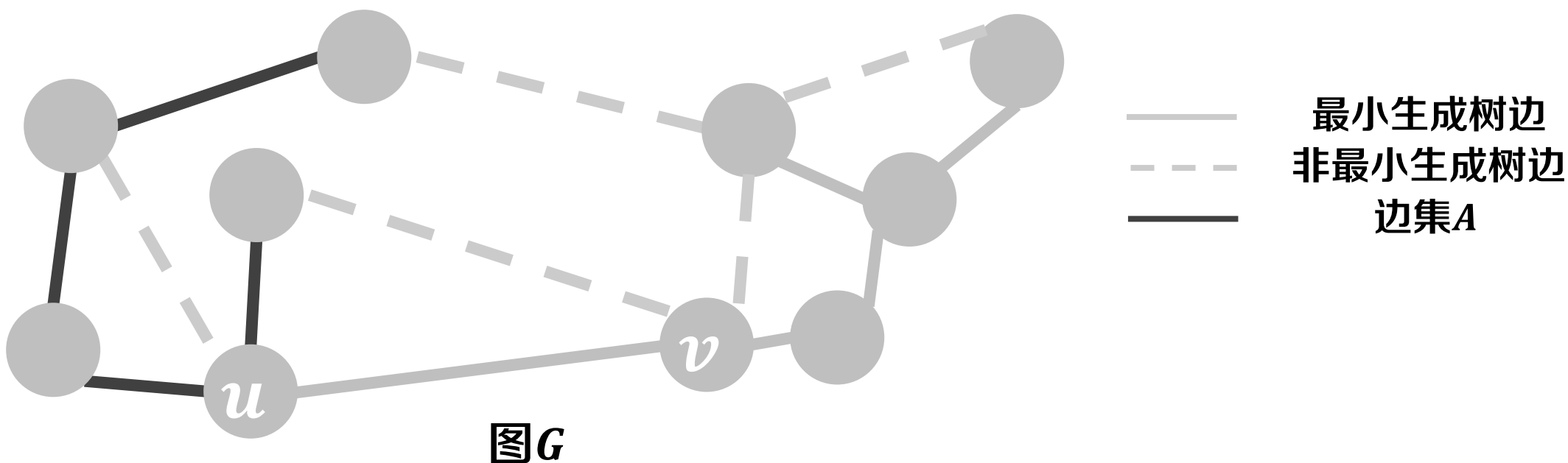
- 给定图 $G = \langle V, E \rangle$ 是一个带权的连通无向图



安全边辨识定理



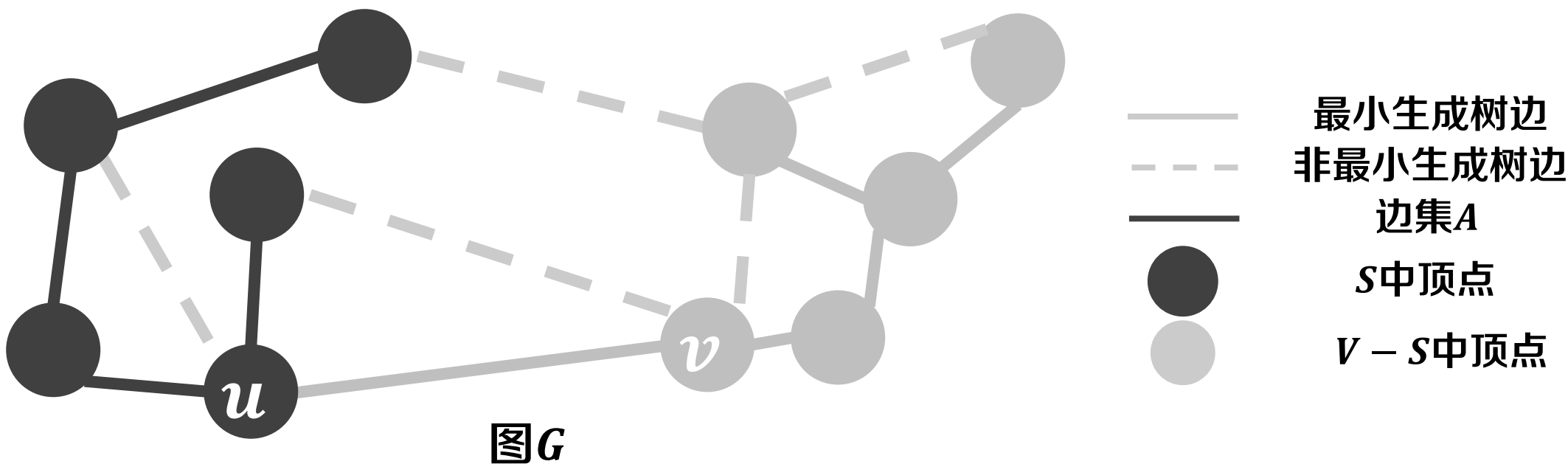
- 给定图 $G = \langle V, E \rangle$ 是一个带权的连通无向图，令 A 为边集 E 的一个子集，且 A 包含在图 G 的某棵最小生成树中



安全边辨识定理



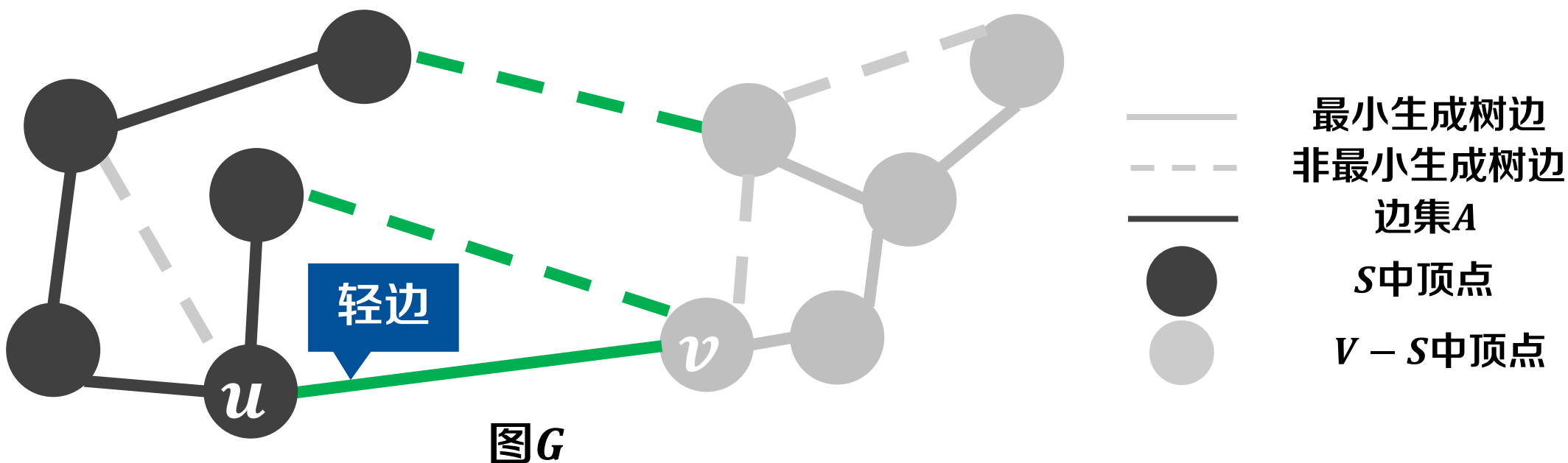
- 给定图 $G = \langle V, E \rangle$ 是一个带权的连通无向图，令 A 为边集 E 的一个子集，且 A 包含在图 G 的某棵最小生成树中
 - 若割 $(S, V - S)$ 是图 G 中 **不妨害边集 A** 的任意割



安全边辨识定理

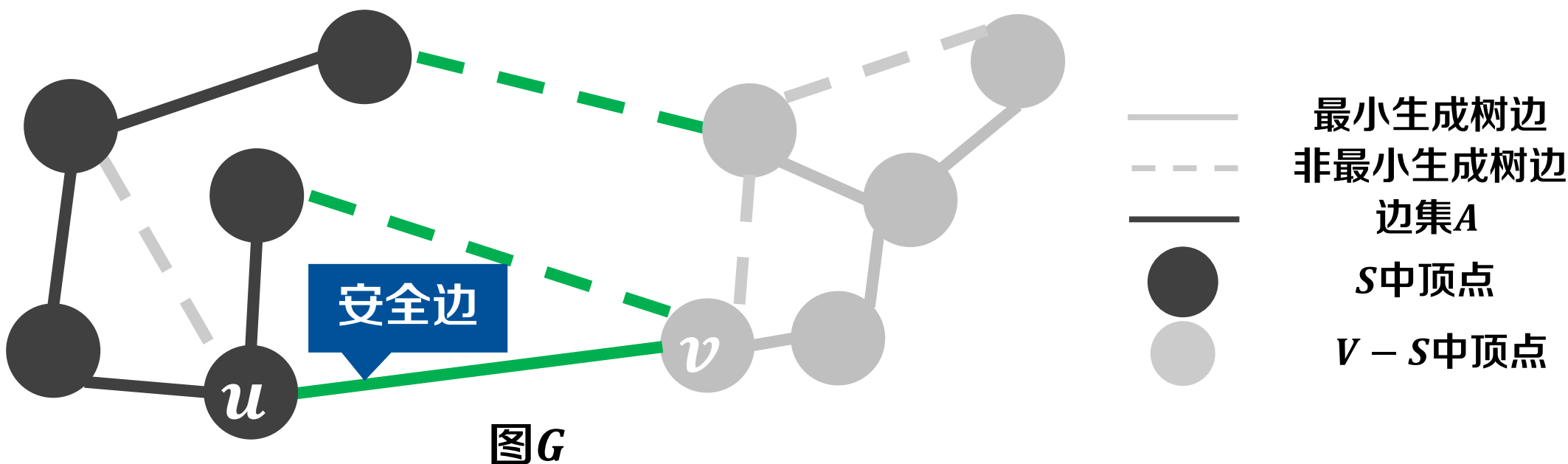


- 给定图 $G = \langle V, E \rangle$ 是一个带权的连通无向图，令 A 为边集 E 的一个子集，且 A 包含在图 G 的某棵最小生成树中
 - 若割 $(S, V - S)$ 是图 G 中不妨害边集 A 的任意割，且 (u, v) 是横跨该割的轻边



安全边辨识定理

- 给定图 $G = \langle V, E \rangle$ 是一个带权的连通无向图，令 A 为边集 E 的一个子集，且 A 包含在图 G 的某棵最小生成树中
 - 若割 $(S, V - S)$ 是图 G 中不妨害边集 A 的任意割，且 (u, v) 是横跨该割的轻边
 - 则对于边集 A ，边 (u, v) 是其安全边

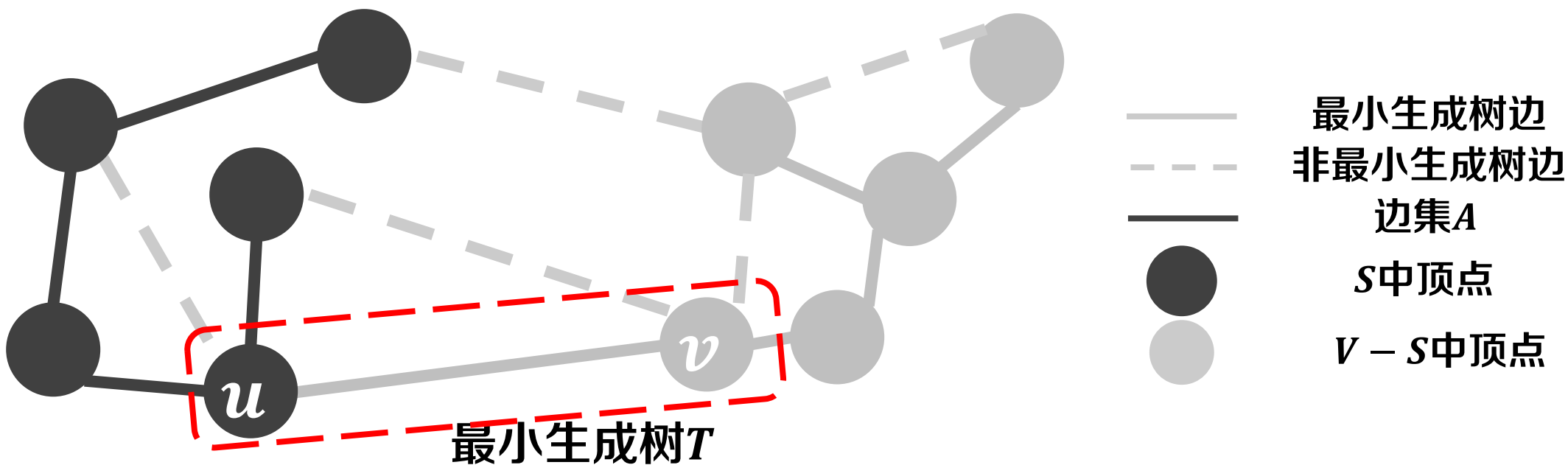


安全边辨识定理



- 证明

- 若 $(u, v) \in T$, 由于 $A \subseteq T$, 则 $A \cup \{(u, v)\} \subseteq T$, 由安全边定义可证

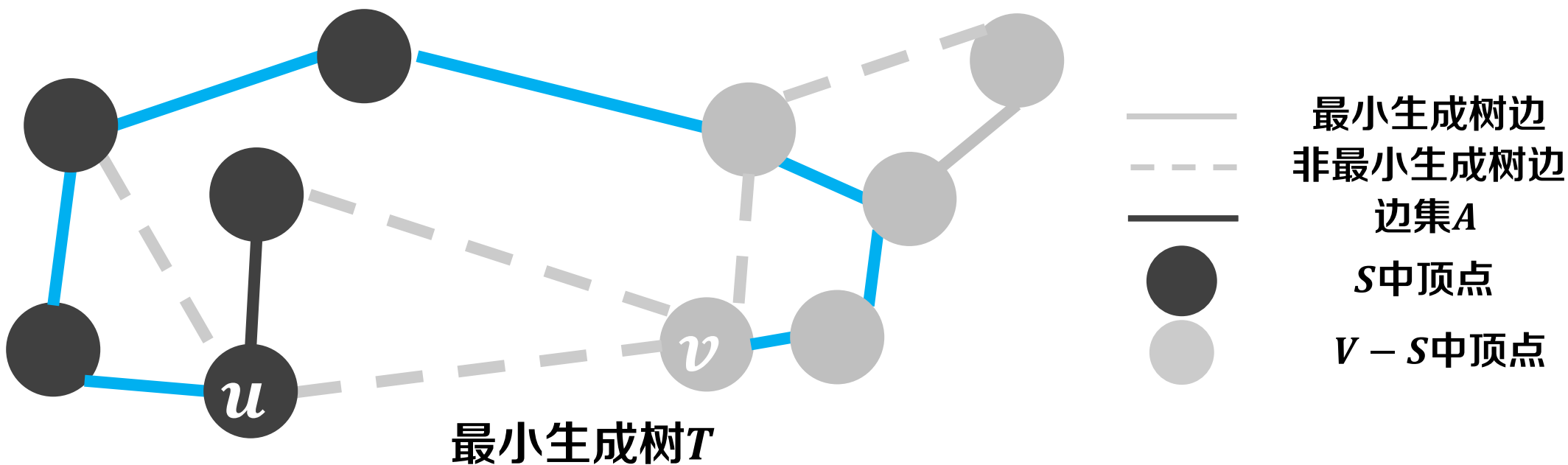


安全边辨识定理



• 证明

- 若 $(u, v) \in T$, 由于 $A \subseteq T$, 则 $A \cup \{(u, v)\} \subseteq T$, 由安全边定义可证
- 若 $(u, v) \notin T$, 则 T 中必存在 u 到 v 的路径 P

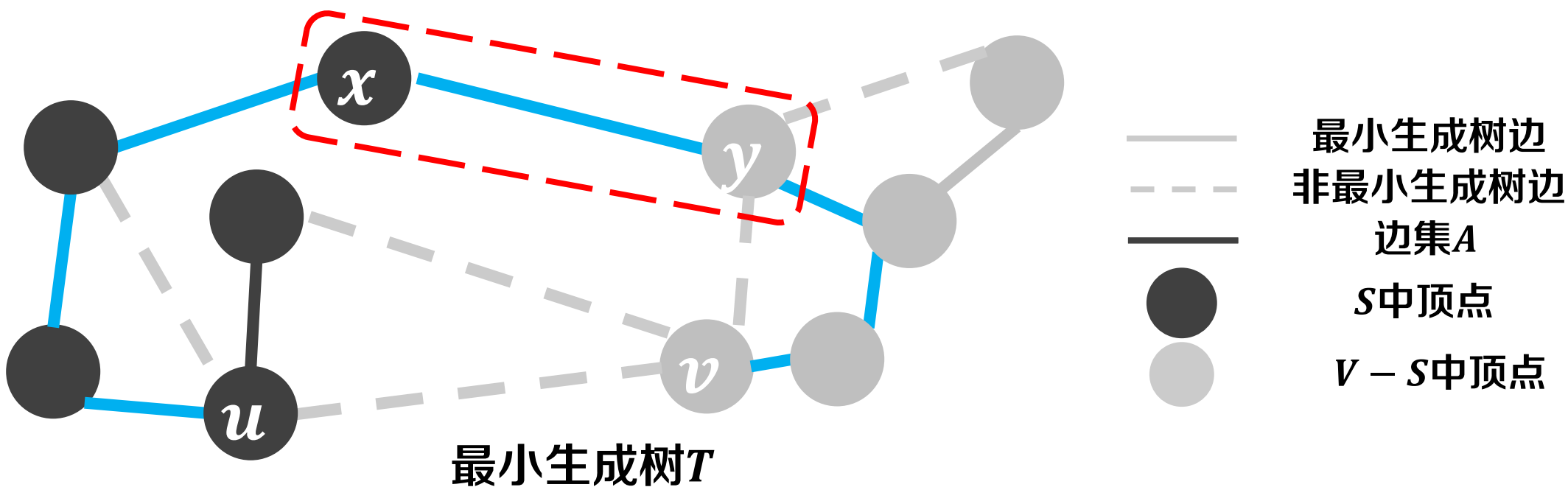


安全边辨识定理



• 证明

- 若 $(u, v) \in T$, 由于 $A \subseteq T$, 则 $A \cup \{(u, v)\} \subseteq T$, 由安全边定义可证
- 若 $(u, v) \notin T$, 则 T 中必存在 u 到 v 的路径 P
 - 不妨设路径 P 中, 横跨割 $(S, V - S)$ 的一条边为 (x, y)

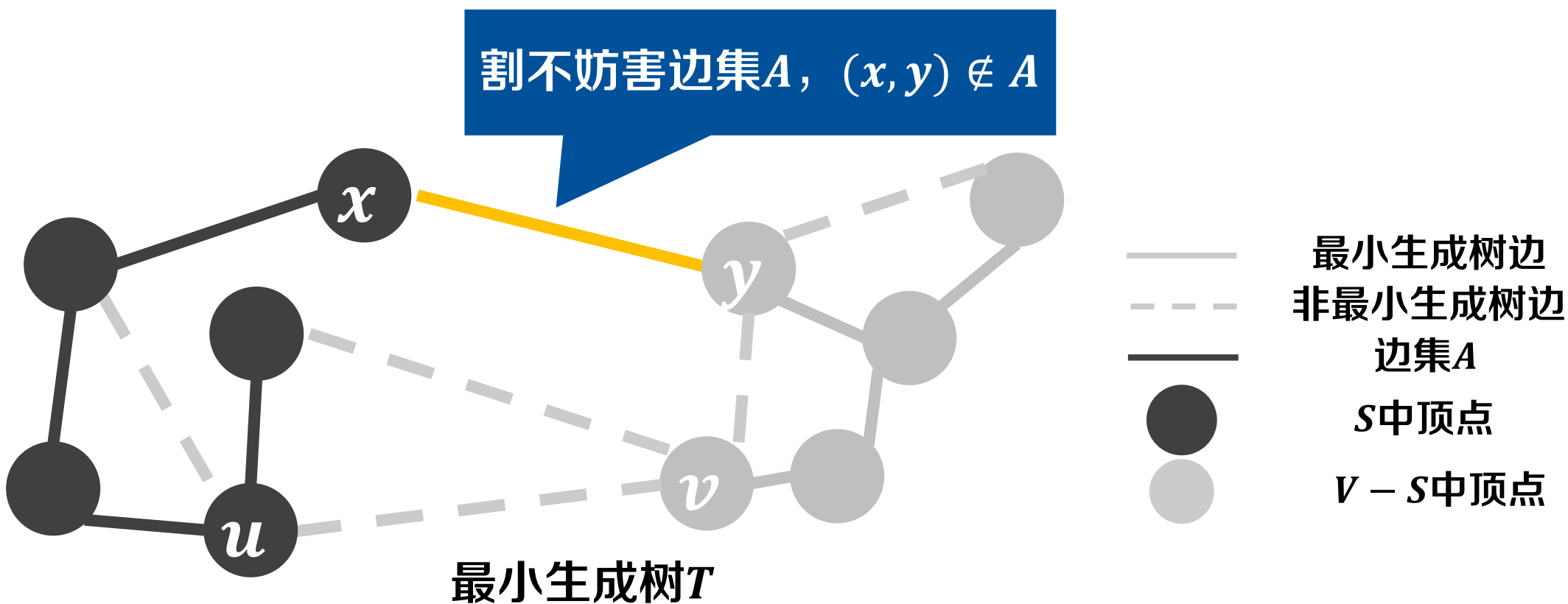


安全边辨识定理



• 证明

- 若 $(u, v) \in T$, 由于 $A \subseteq T$, 则 $A \cup \{(u, v)\} \subseteq T$, 由安全边定义可证
- 若 $(u, v) \notin T$, 则 T 中必存在 u 到 v 的路径 P
 - 不妨设路径 P 中, 横跨割 $(S, V - S)$ 的一条边为 (x, y)

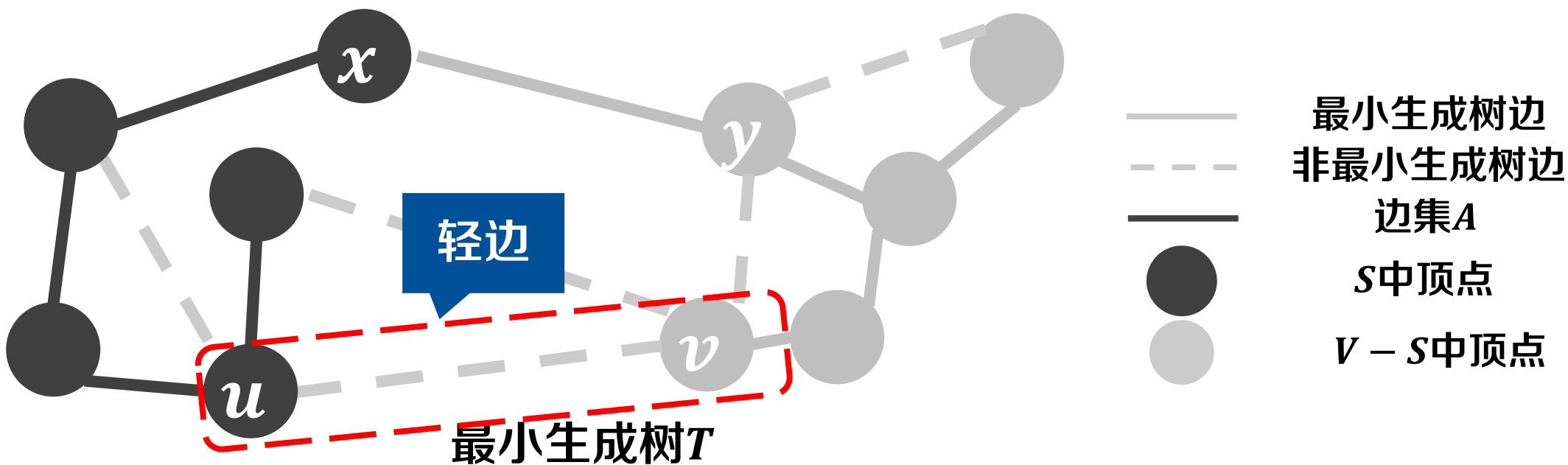


安全边辨识定理



• 证明

- 若 $(u, v) \in T$, 由于 $A \subseteq T$, 则 $A \cup \{(u, v)\} \subseteq T$, 由安全边定义可证
- 若 $(u, v) \notin T$, 则 T 中必存在 u 到 v 的路径 P
 - 不妨设路径 P 中, 横跨割 $(S, V - S)$ 的一条边为 (x, y)
 - 边 (u, v) 是横跨割的轻边, 所以 $w(u, v) \leq w(x, y)$

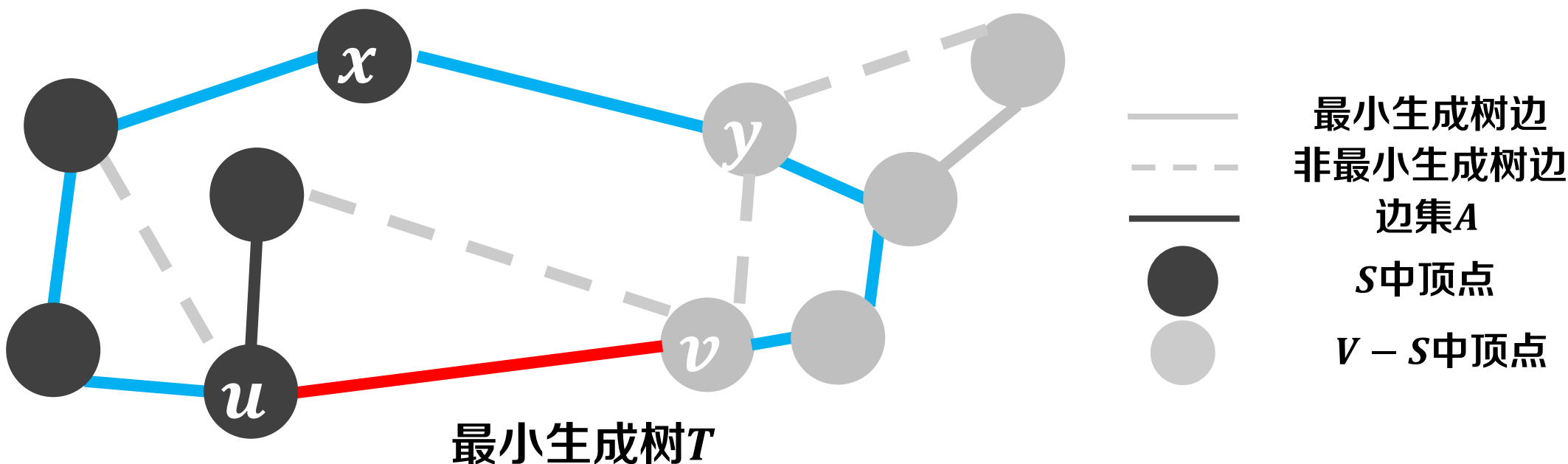


安全边辨识定理



• 证明

- 若 $(u, v) \in T$, 由于 $A \subseteq T$, 则 $A \cup \{(u, v)\} \subseteq T$, 由安全边定义可证
- 若 $(u, v) \notin T$, 则 T 中必存在 u 到 v 的路径 P
 - 不妨设路径 P 中, 横跨割 $(S, V - S)$ 的一条边为 (x, y)
 - 边 (u, v) 是横跨割的轻边, 所以 $w(u, v) \leq w(x, y)$
 - 将边 (u, v) 加入到 T 中会形成环路

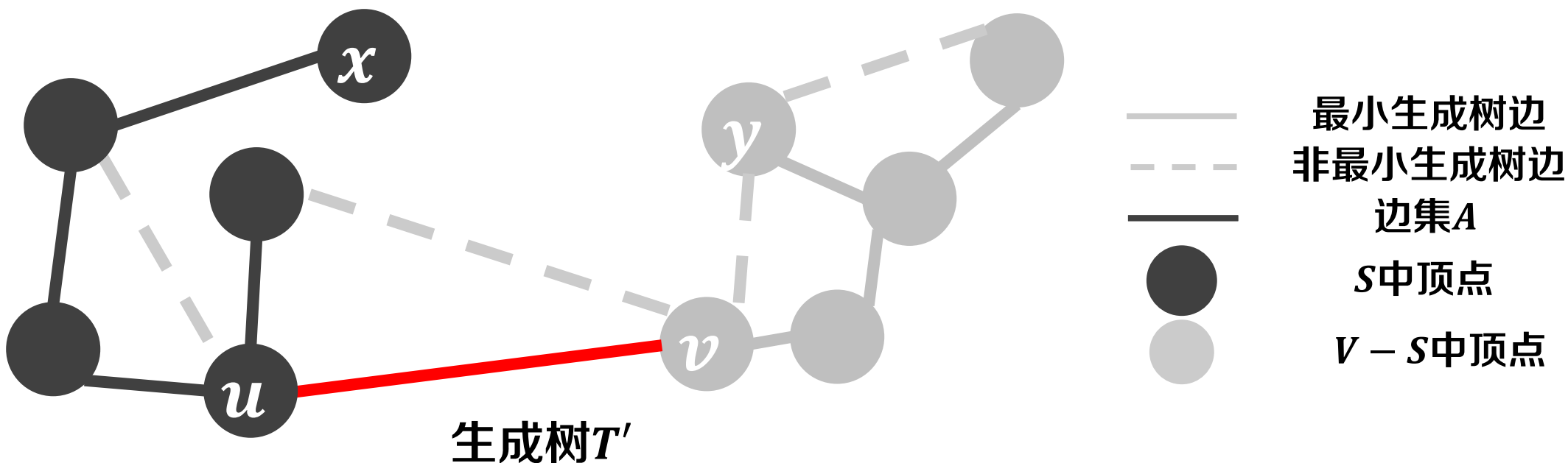


安全边辨识定理



• 证明

- 若 $(u, v) \in T$, 由于 $A \subseteq T$, 则 $A \cup \{(u, v)\} \subseteq T$, 由安全边定义可证
- 若 $(u, v) \notin T$, 则 T 中必存在 u 到 v 的路径 P
 - 不妨设路径 P 中, 横跨割 $(S, V - S)$ 的一条边为 (x, y)
 - 边 (u, v) 是横跨割的轻边, 所以 $w(u, v) \leq w(x, y)$
 - 将边 (u, v) 加入到 T 中会形成环路, 再去掉边 (x, y) 会形成另一棵树 T'

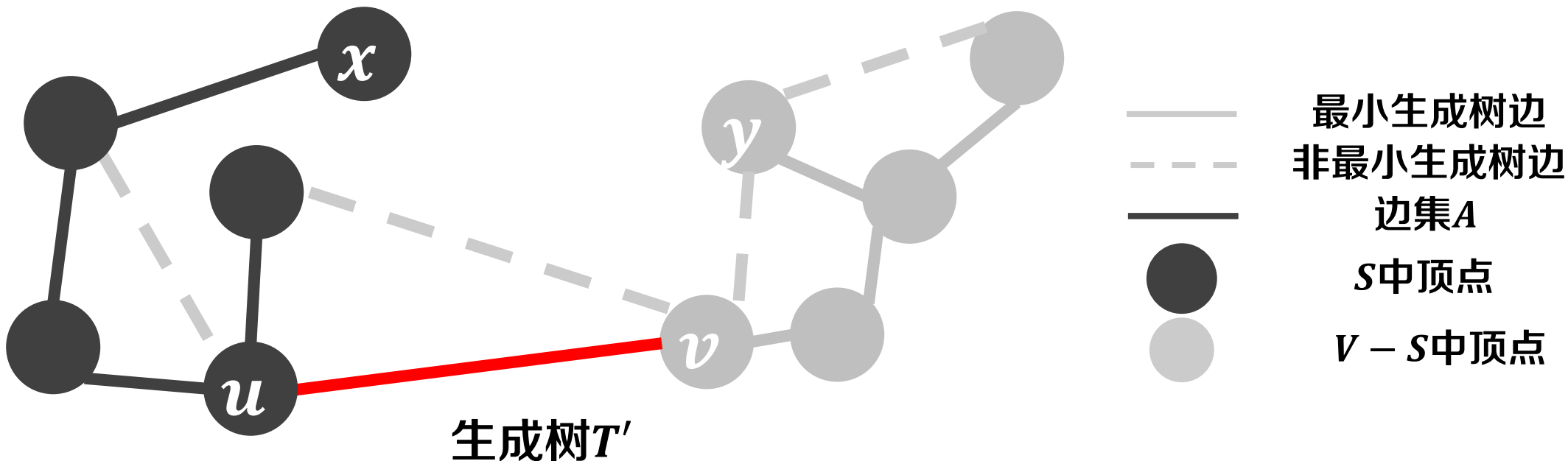


安全边辨识定理



• 证明

- 若 $(u, v) \in T$, 由于 $A \subseteq T$, 则 $A \cup \{(u, v)\} \subseteq T$, 由安全边定义可证
- 若 $(u, v) \notin T$, 则 T 中必存在 u 到 v 的路径 P
 - 不妨设路径 P 中, 横跨割 $(S, V - S)$ 的一条边为 (x, y)
 - 边 (u, v) 是横跨割的轻边, 所以 $w(u, v) \leq w(x, y)$
 - 将边 (u, v) 加入到 T 中会形成环路, 再去掉边 (x, y) 会形成另一棵树 T'
 - $w(T') = w(T) + w(u, v) - w(x, y)$

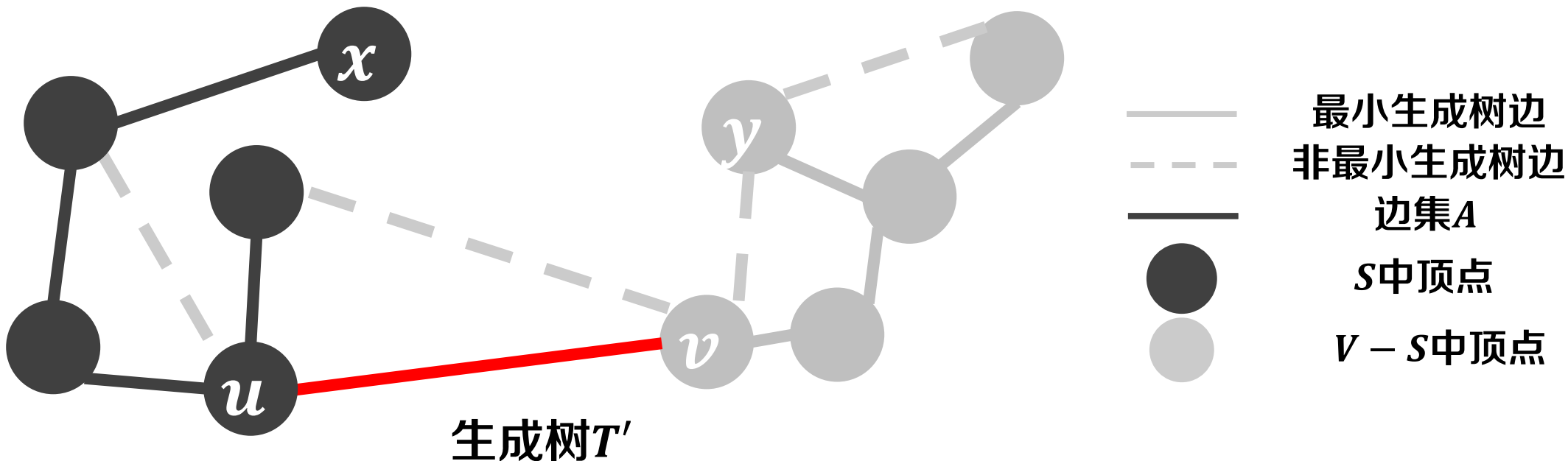


安全边辨识定理



• 证明

- 若 $(u, v) \in T$, 由于 $A \subseteq T$, 则 $A \cup \{(u, v)\} \subseteq T$, 由安全边定义可证
- 若 $(u, v) \notin T$, 则 T 中必存在 u 到 v 的路径 P
 - 不妨设路径 P 中, 横跨割 $(S, V - S)$ 的一条边为 (x, y)
 - 边 (u, v) 是横跨割的轻边, 所以 $w(u, v) \leq w(x, y)$
 - 将边 (u, v) 加入到 T 中会形成环路, 再去掉边 (x, y) 会形成另一棵树 T'
 - $w(T') = w(T) + w(u, v) - w(x, y) \leq w(T)$

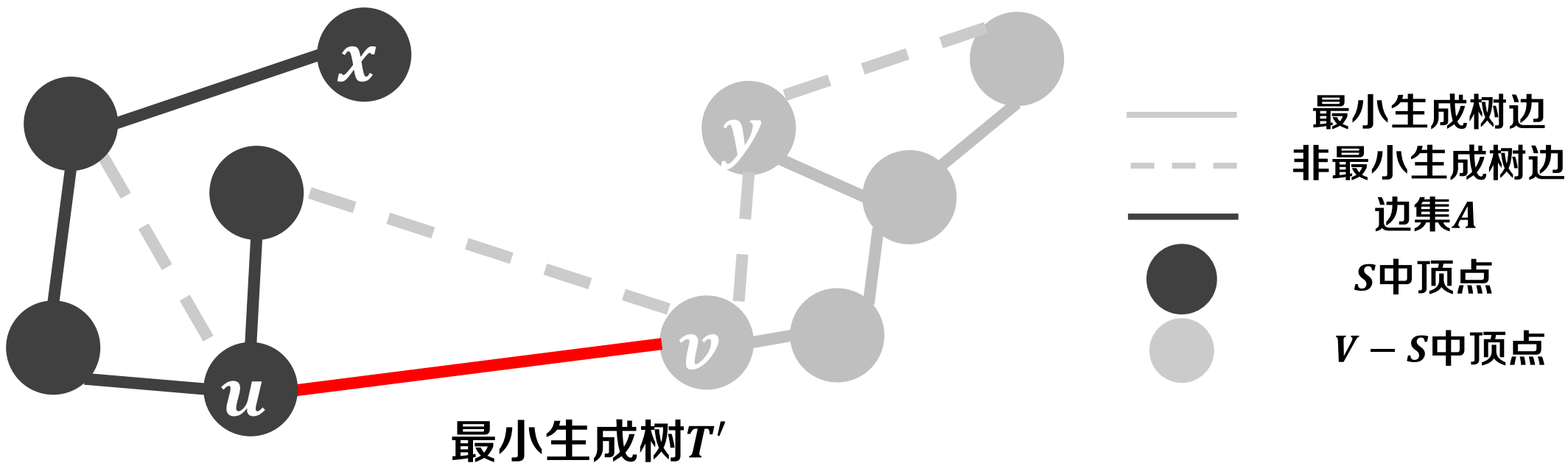


安全边辨识定理



• 证明

- 若 $(u, v) \in T$, 由于 $A \subseteq T$, 则 $A \cup \{(u, v)\} \subseteq T$, 由安全边定义可证
- 若 $(u, v) \notin T$, 则 T 中必存在 u 到 v 的路径 P
 - 不妨设路径 P 中, 横跨割 $(S, V - S)$ 的一条边为 (x, y)
 - 边 (u, v) 是横跨割的轻边, 所以 $w(u, v) \leq w(x, y)$
 - 将边 (u, v) 加入到 T 中会形成环路, 再去掉边 (x, y) 会形成另一棵树 T'
 - $w(T') \leq w(T)$, T' 也是最小生成树

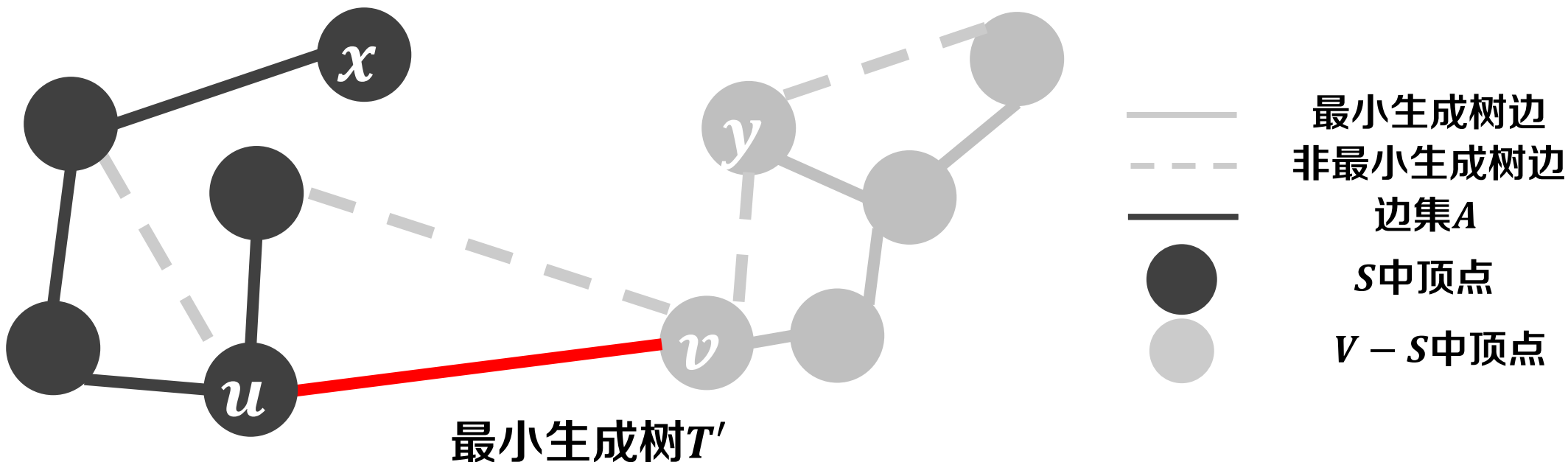


安全边辨识定理



• 证明

- 若 $(u, v) \in T$, 由于 $A \subseteq T$, 则 $A \cup \{(u, v)\} \subseteq T$, 由安全边定义可证
- 若 $(u, v) \notin T$, 则 T 中必存在 u 到 v 的路径 P
 - 不妨设路径 P 中, 横跨割 $(S, V - S)$ 的一条边为 (x, y)
 - 边 (u, v) 是横跨割的轻边, 所以 $w(u, v) \leq w(x, y)$
 - 将边 (u, v) 加入到 T 中会形成环路, 再去掉边 (x, y) 会形成另一棵树 T'
 - $w(T') \leq w(T)$, T' 也是最小生成树, $A \cup \{(u, v)\} \subseteq T'$, 边 (u, v) 是**安全边**



- 生成树是一个连通、无环的生成子图
 - 新建一个空边集 A ，边集 A 可逐步扩展为最小生成树
 - 每次向边集 A 中新增加一条边
 - 需保证边集 A 仍是一个无环图
 - 需保证边集 A 仍是最小生成树的子集

- 生成树是一个连通、无环的生成子图
 - 新建一个空边集 A ，边集 A 可逐步扩展为最小生成树
 - 每次向边集 A 中新增加一条边
 - 需保证边集 A 仍是一个无环图
 - 需保证边集 A 仍是最小生成树的子集

添加一条轻边

- 生成树是一个连通、无环的生成子图
 - 新建一个空边集 A ，边集 A 可逐步扩展为最小生成树
 - 每次向边集 A 中新增加一条边
 - 需保证边集 A 仍是一个无环图
 - 需保证边集 A 仍是最小生成树的子集

添加一条轻边

问题：如何有效地实现此贪心策略？

通用框架

- 生成树是一个连通、无环的生成子图
 - 新建一个空边集 A ，边集 A 可逐步扩展为最小生成树
 - 每次向边集 A 中新增加一条边
 - 需保证边集 A 仍是一个无环图
 - 需保证边集 A 仍是最小生成树的子集

添加一条轻边

问题：如何有效地实现此贪心策略？

Prim算法

Kruskal算法

通用框架

- 生成树是一个连通、无环的生成子图
 - 新建一个空边集 A ，边集 A 可逐步扩展为最小生成树
 - 每次向边集 A 中新增加一条边
 - 需保证边集 A 仍是一个无环图
 - 需保证边集 A 仍是最小生成树的子集

添加一条轻边

问题：如何有效地实现此贪心策略？

Prim算法

Kruskal算法

问题背景

通用框架

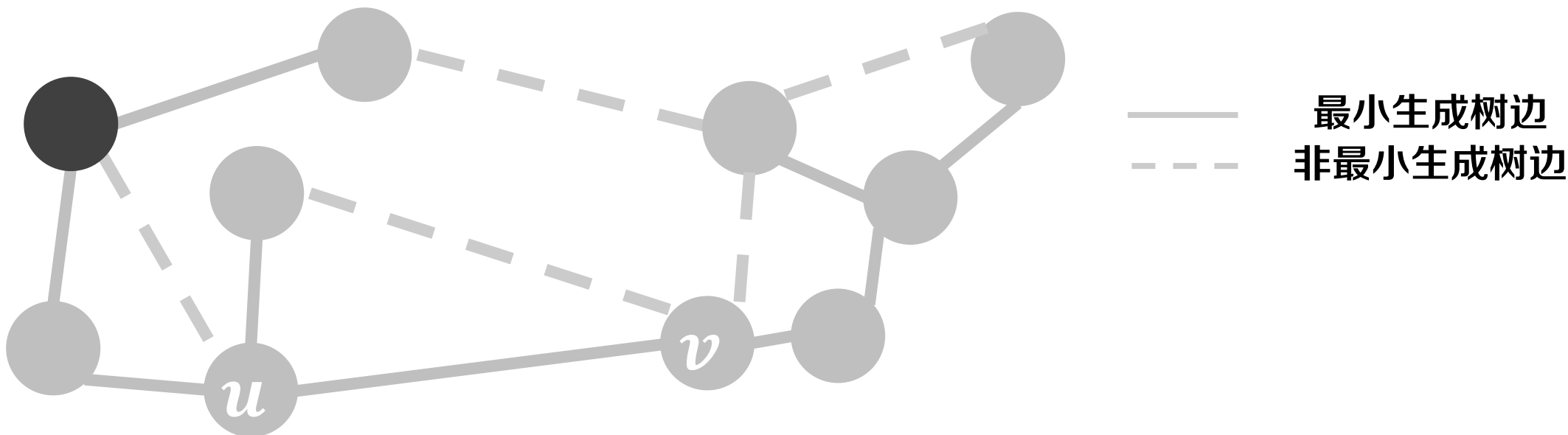
Prim算法

算法实例

算法分析

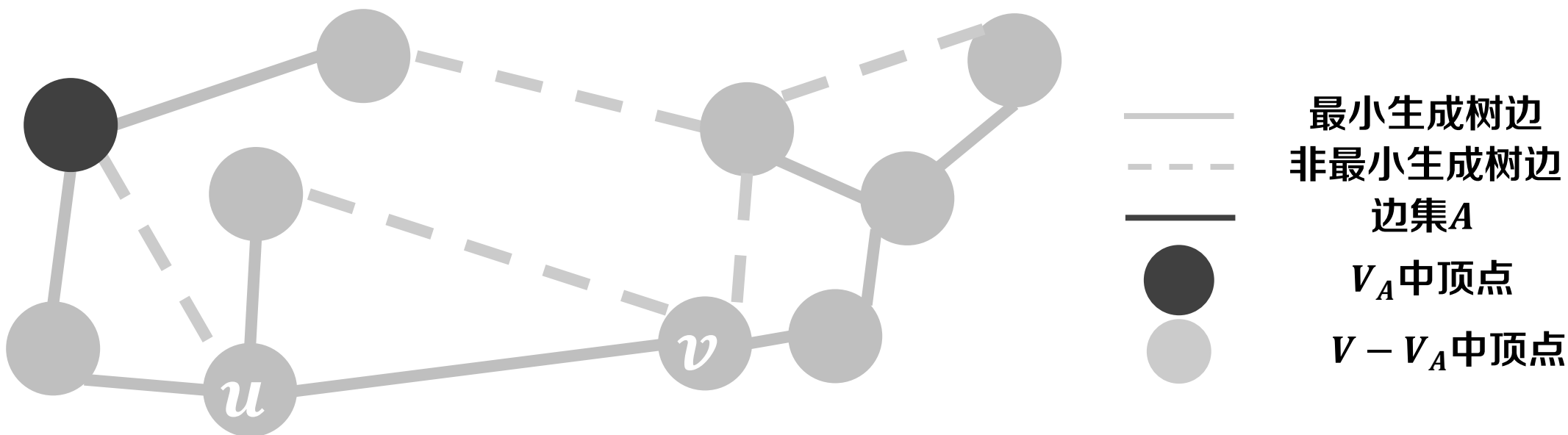
- 算法思想

- 步骤1: 选择任意一个顶点, 作为生成树的起始顶点



- 算法思想

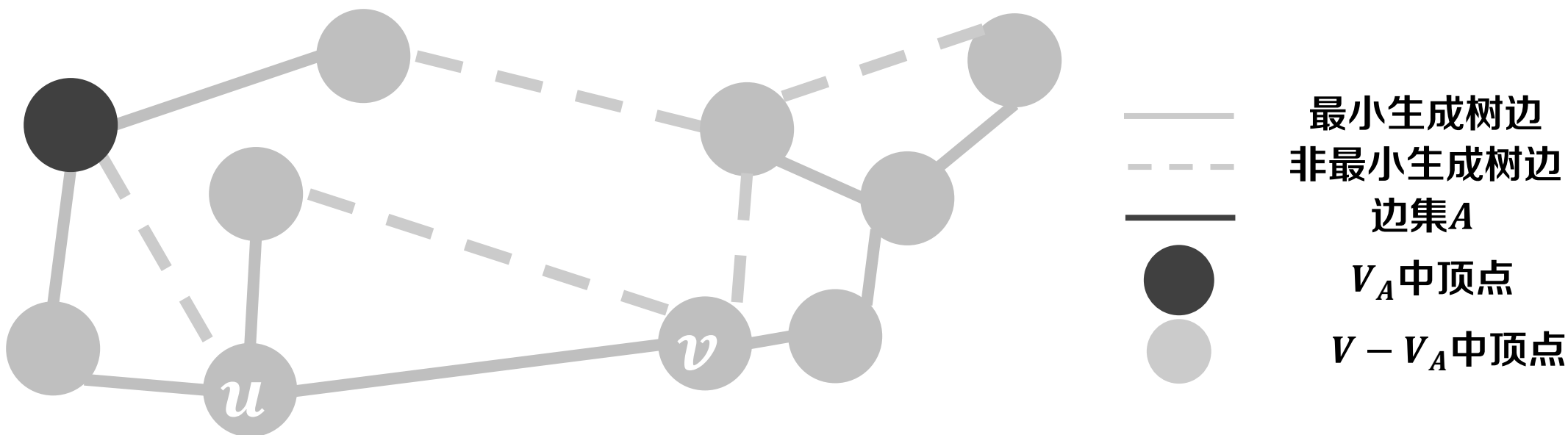
- 步骤1: 选择任意一个顶点, 作为生成树的起始顶点
- 步骤2: 保持边集 A 始终为一棵树



- 算法思想

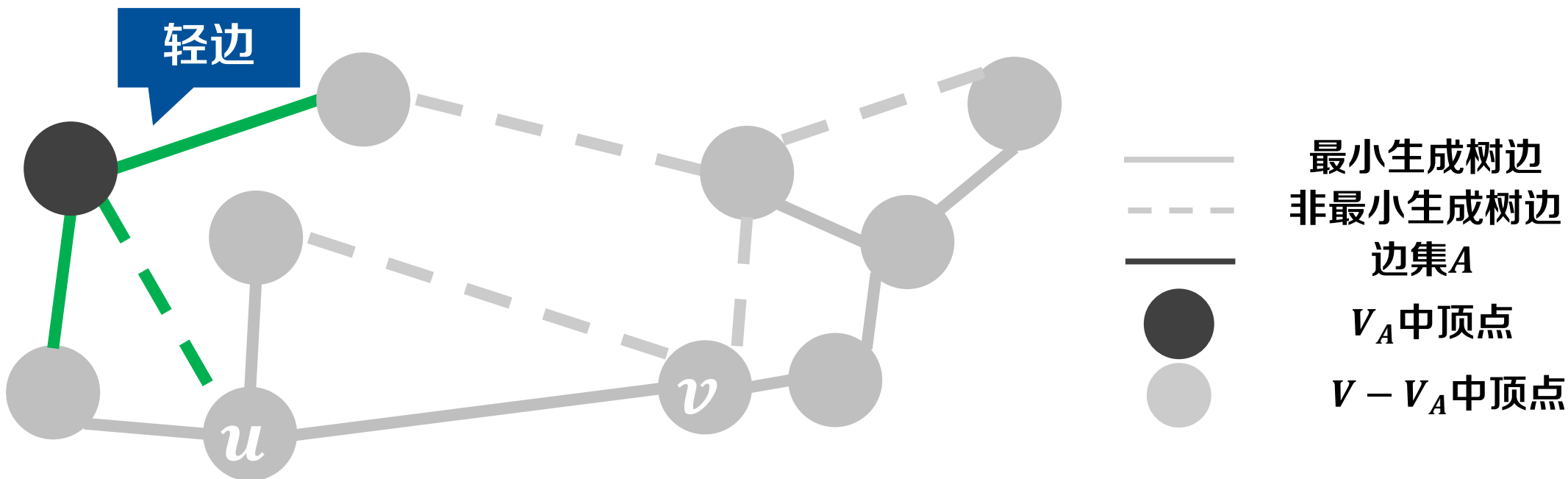
- 步骤1: 选择任意一个顶点, 作为生成树的起始顶点
- 步骤2: 保持边集 A 始终为一棵树, 选择割 $(V_A, V - V_A)$

树中顶点



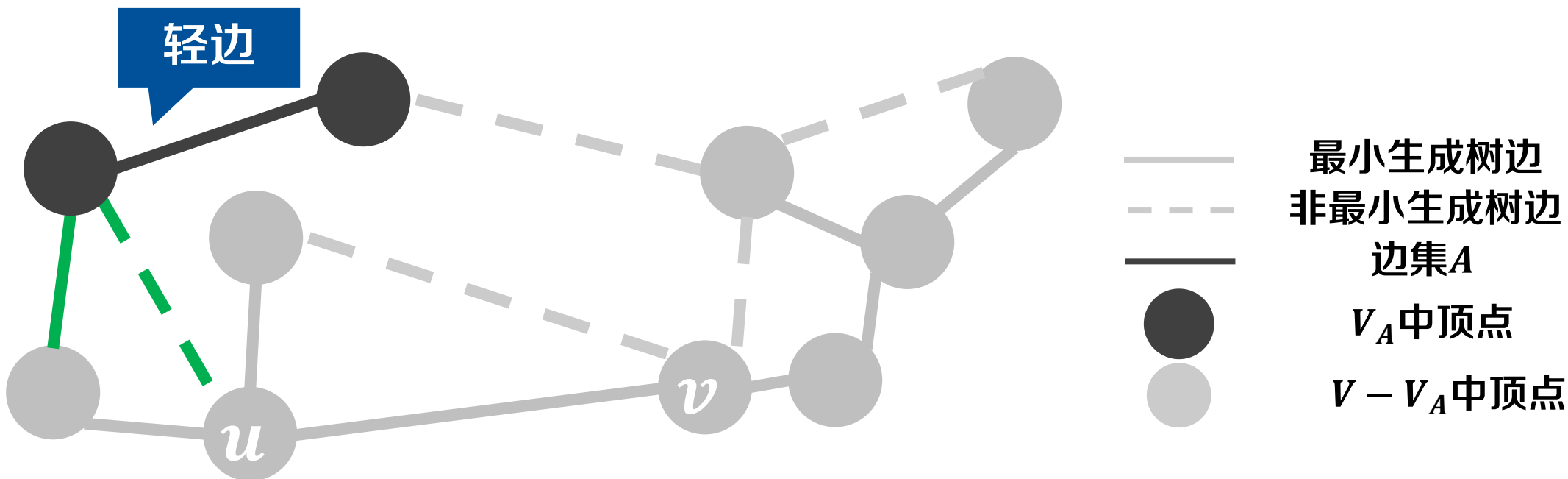
● 算法思想

- 步骤1: 选择任意一个顶点, 作为生成树的起始顶点
- 步骤2: 保持边集 A 始终为一棵树, 选择割 $(V_A, V - V_A)$
- 步骤3: 选择横跨割 $(V_A, V - V_A)$ 的轻边



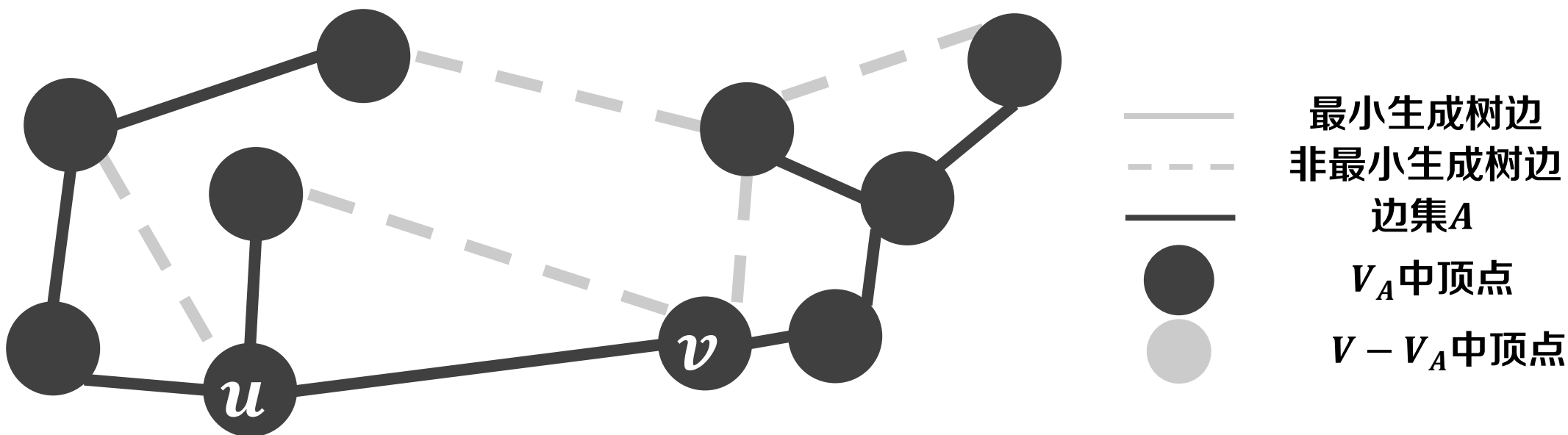
• 算法思想

- 步骤1: 选择任意一个顶点, 作为生成树的起始顶点
- 步骤2: 保持边集 A 始终为一棵树, 选择割 $(V_A, V - V_A)$
- 步骤3: 选择横跨割 $(V_A, V - V_A)$ 的轻边, 添加到边集 A 中



• 算法思想

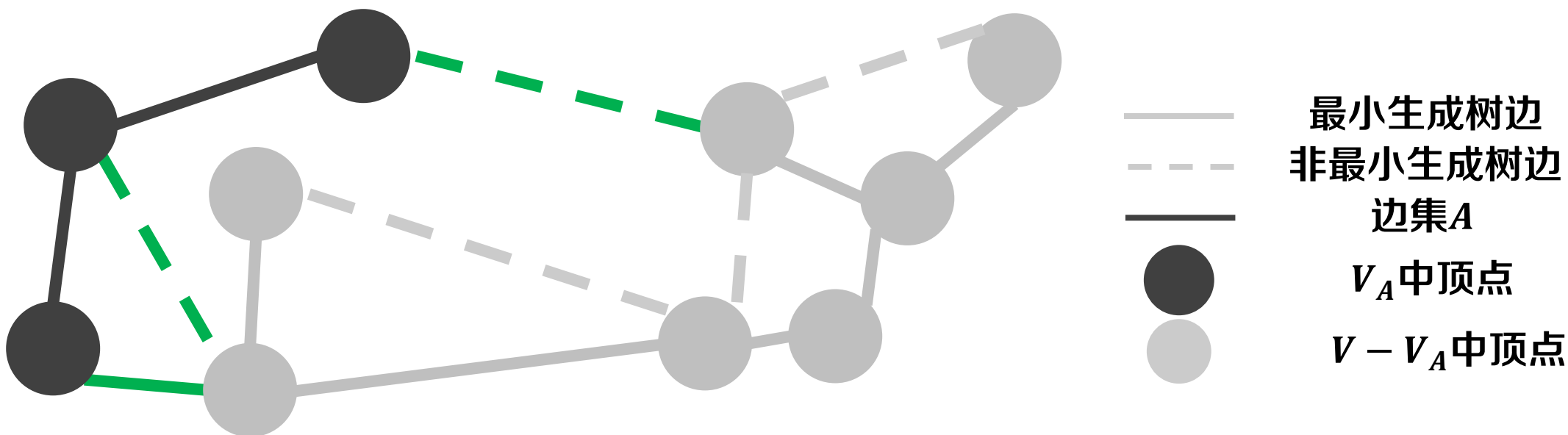
- 步骤1: 选择任意一个顶点, 作为生成树的起始顶点
- 步骤2: 保持边集 A 始终为一棵树, 选择割 $(V_A, V - V_A)$
- 步骤3: 选择横跨割 $(V_A, V - V_A)$ 的轻边, 添加到边集 A 中
- 步骤4: 重复步骤2和步骤3, 直至覆盖所有顶点



- 辅助数组

- *color*表示顶点状态

- 黑色顶点 u 已覆盖, $u \in V_A$
 - 白色顶点 u 未覆盖, $u \in V - V_A$



Prim算法



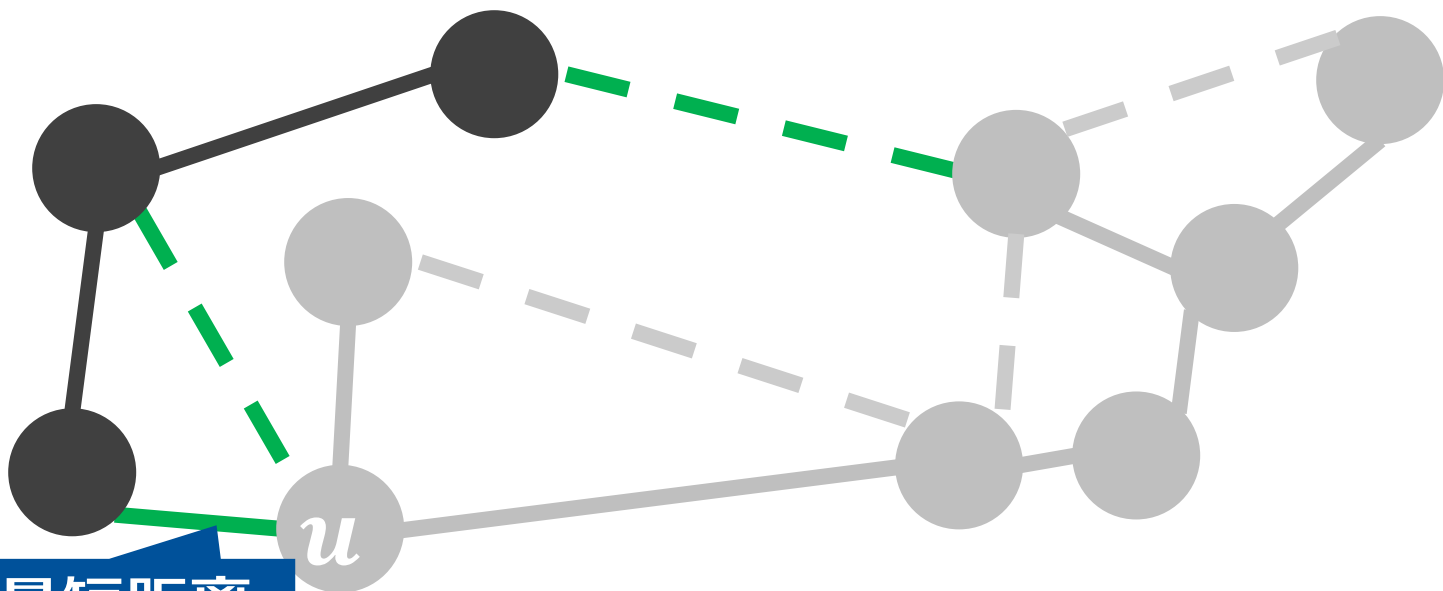
- 辅助数组

- $color$ 表示顶点状态

- 黑色顶点 u 已覆盖, $u \in V_A$
 - 白色顶点 u 未覆盖, $u \in V - V_A$

- $dist$ 记录横跨 $(V_A, V - V_A)$ 边的权重

- 顶点集 V_A 到顶点 u 的最短距离, $dist[u] = \min\{w(x, u)\}, \forall x \in V_A$



最小生成树边
非最小生成树边
边集A

V_A 中顶点

$V - V_A$ 中顶点

到 u 最短距离

Prim算法



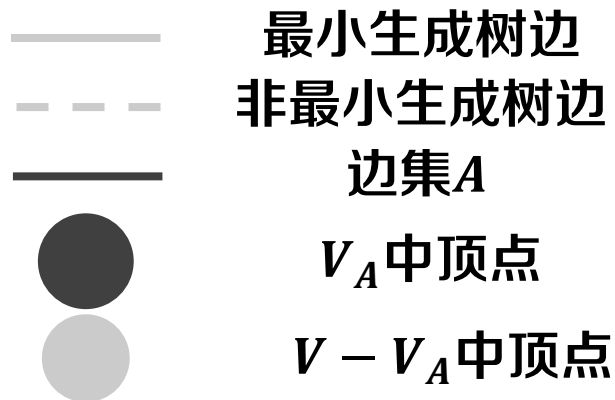
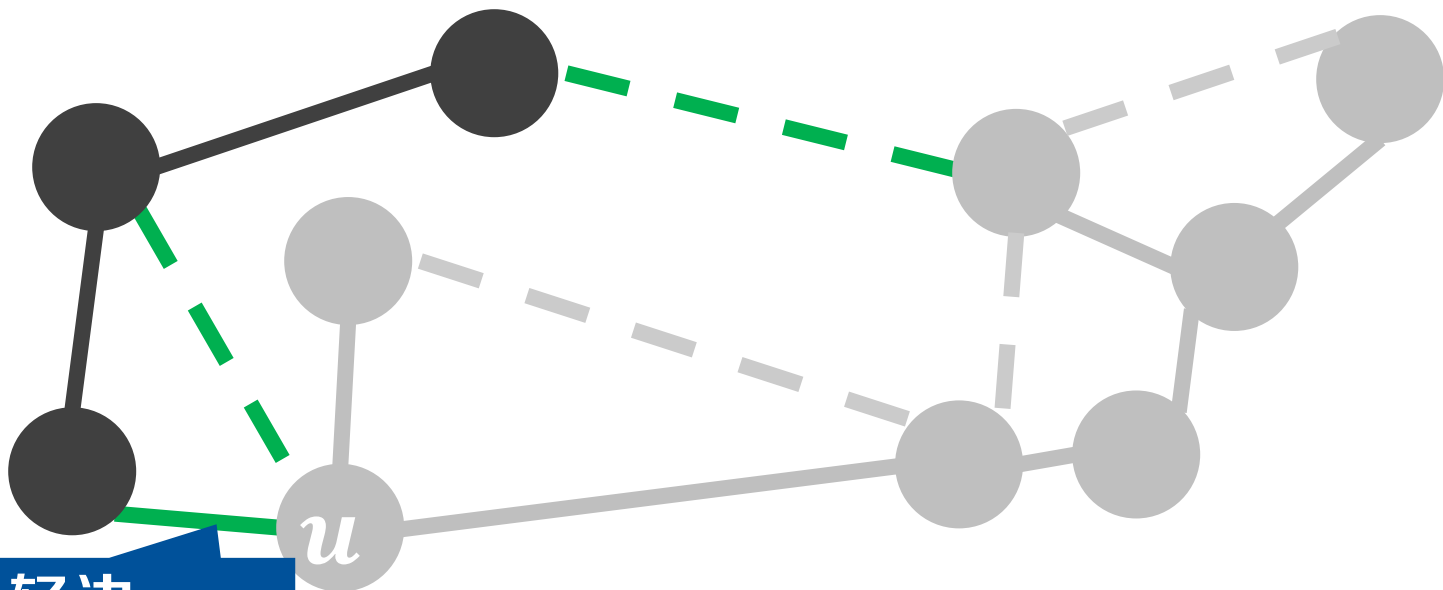
- 辅助数组

- *color*表示顶点状态

- 黑色顶点 u 已覆盖, $u \in V_A$
 - 白色顶点 u 未覆盖, $u \in V - V_A$

- *dist*记录横跨 $(V_A, V - V_A)$ 边的权重

- 顶点集 V_A 到顶点 u 的最短距离, $dist[u] = \min\{w(x, u)\}, \forall x \in V_A$
 - **轻边**: $\min\{dist[u]\}, \forall u \in V - V_A$



轻边

- 辅助数组

- *color*表示顶点状态

- 黑色顶点 u 已覆盖, $u \in V_A$
 - 白色顶点 u 未覆盖, $u \in V - V_A$

- *dist*记录横跨($V_A, V - V_A$)边的权重

- 顶点集 V_A 到顶点 u 的最短距离, $dist[u] = \min\{w(x, u)\}, \forall x \in V_A$
 - **轻边**: $\min\{dist[u]\}, \forall u \in V - V_A$

- *pred*表示前驱顶点

- $(pred[u], u)$ 为最小生成树的边

问题背景

通用框架

Prim算法

算法实例

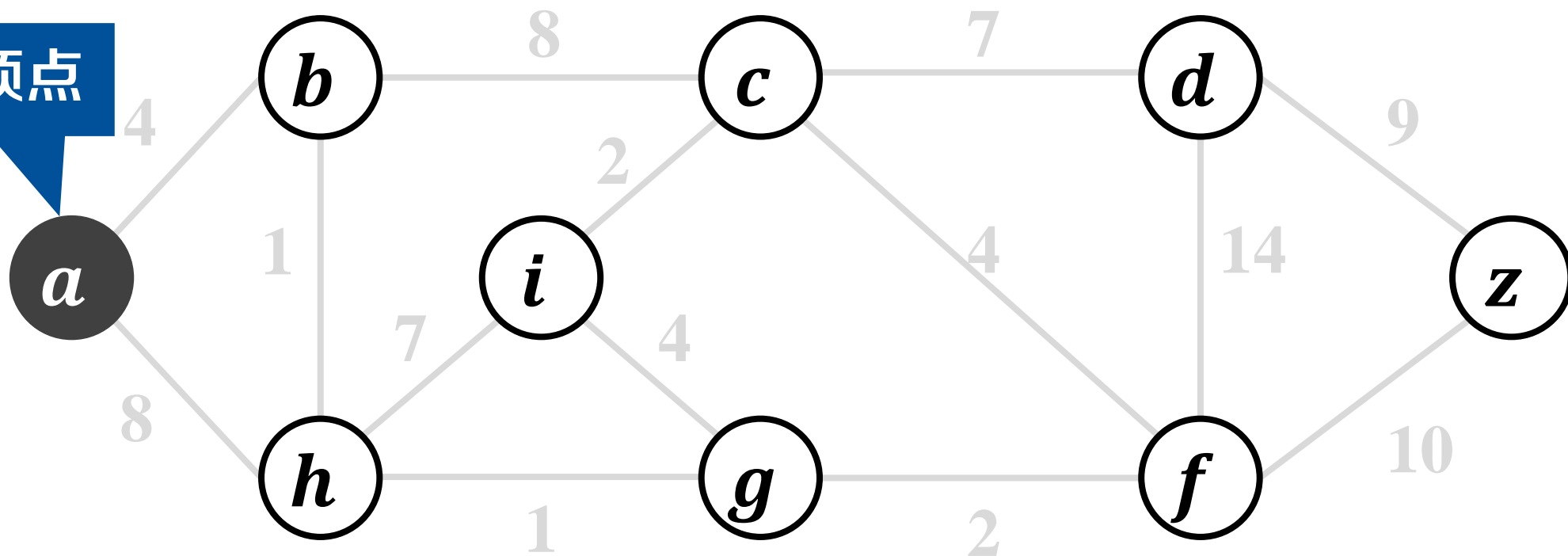
算法分析

算法实例



<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	W	W	W	W	W	W	W	W	W
<i>dist</i>	∞	∞	∞	∞	∞	∞	∞	∞	∞
<i>pred</i>	N	N	N	N	N	N	N	N	N

起始顶点

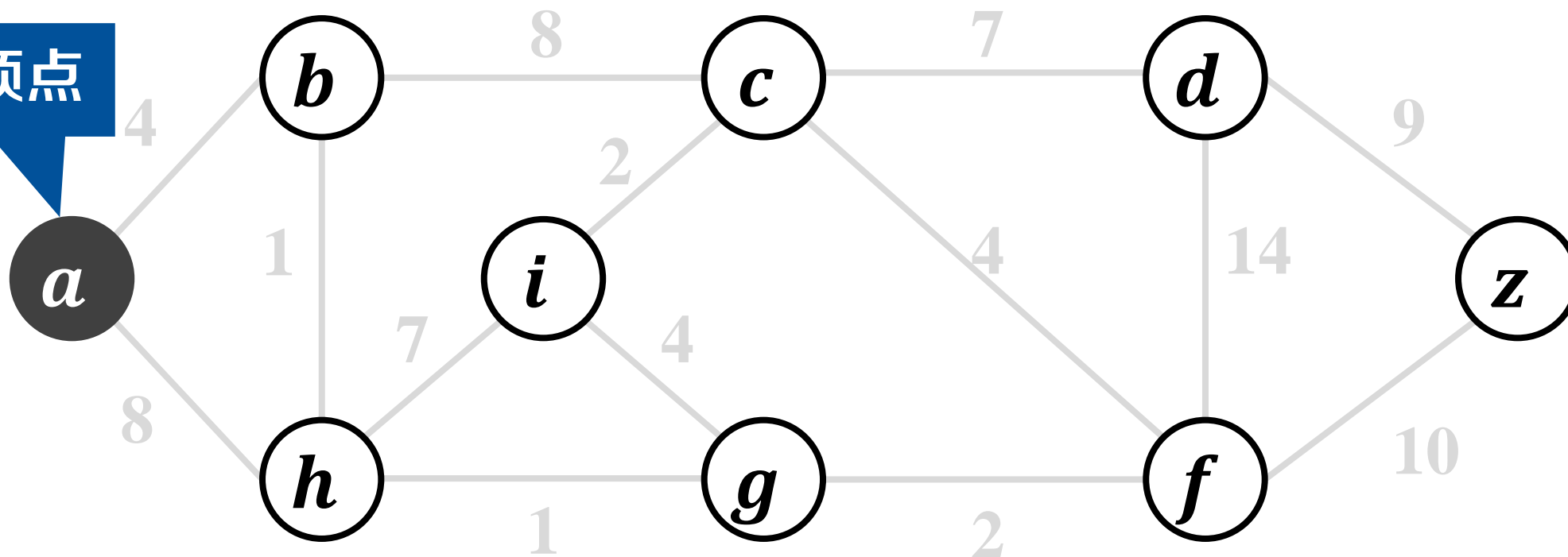


算法实例



<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	W	W	W	W	W	W	W	W
<i>dist</i>	0	∞	∞	∞	∞	∞	∞	∞	∞
<i>pred</i>	N	N	N	N	N	N	N	N	N

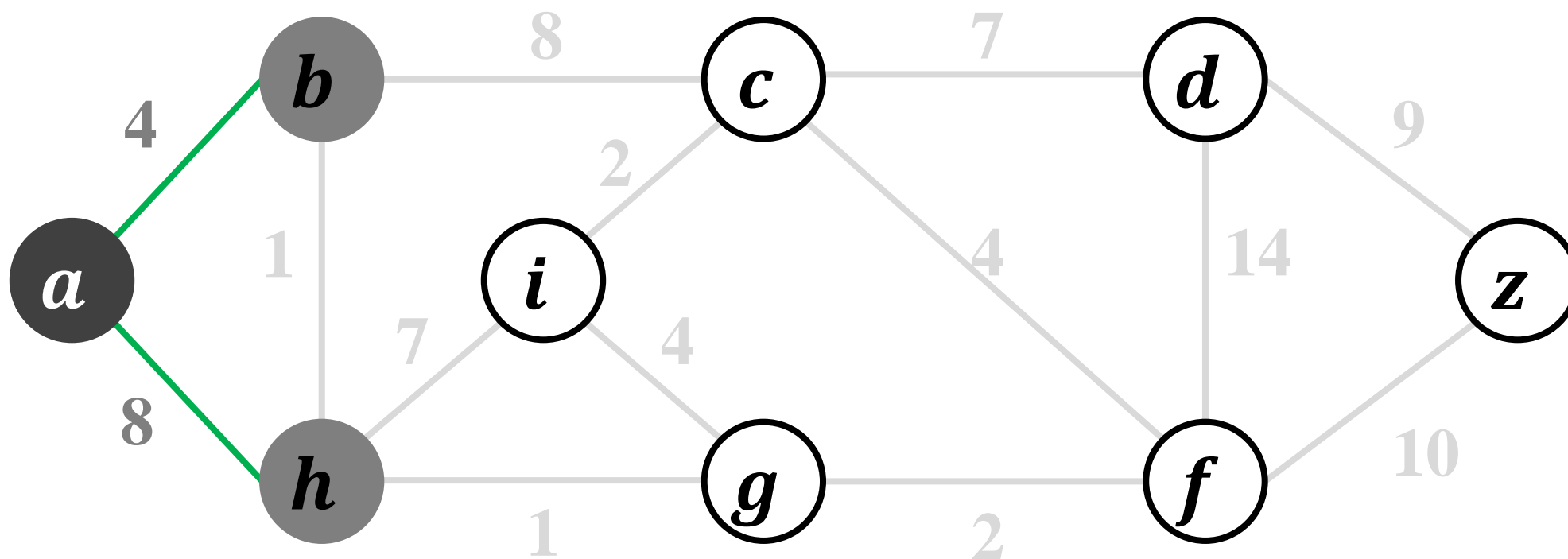
起始顶点



算法实例



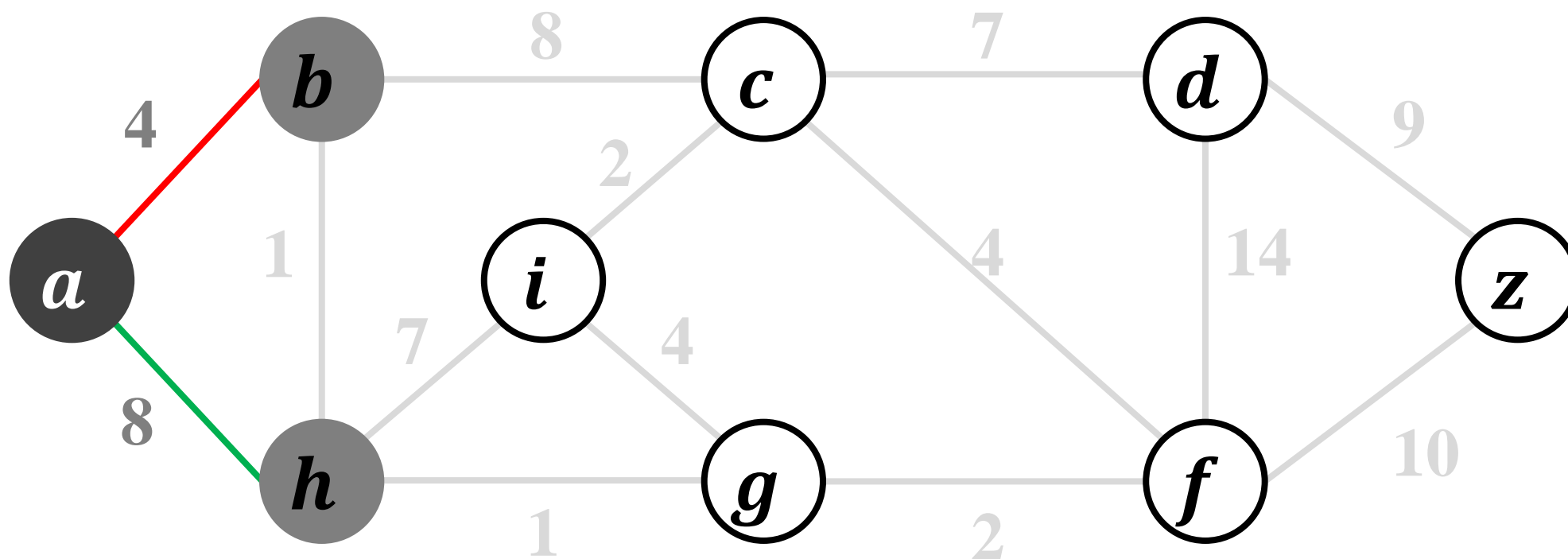
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	W	W	W	W	W	W	W	W
<i>dist</i>	0	4	∞	∞	∞	∞	8	∞	∞
<i>pred</i>	N	<i>a</i>	N	N	N	N	<i>a</i>	N	N



算法实例



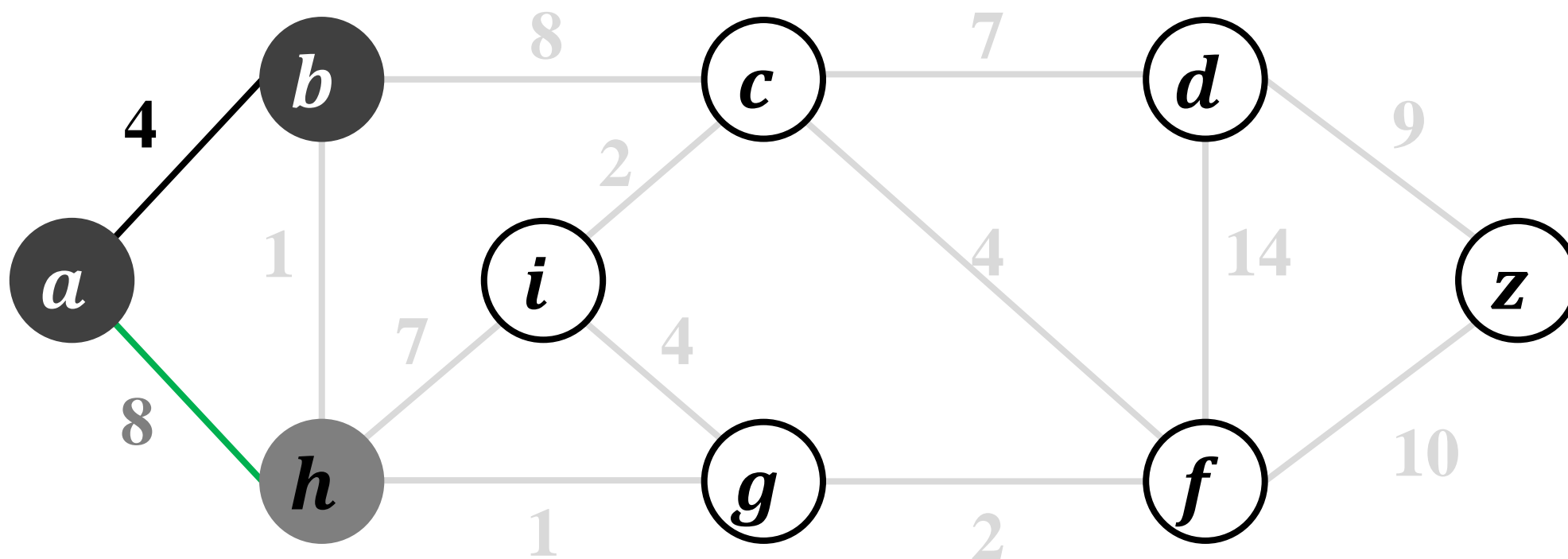
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	W	W	W	W	W	W	W	W
<i>dist</i>	0	4	∞	∞	∞	∞	8	∞	∞
<i>pred</i>	N	<i>a</i>	N	N	N	N	<i>a</i>	N	N



算法实例



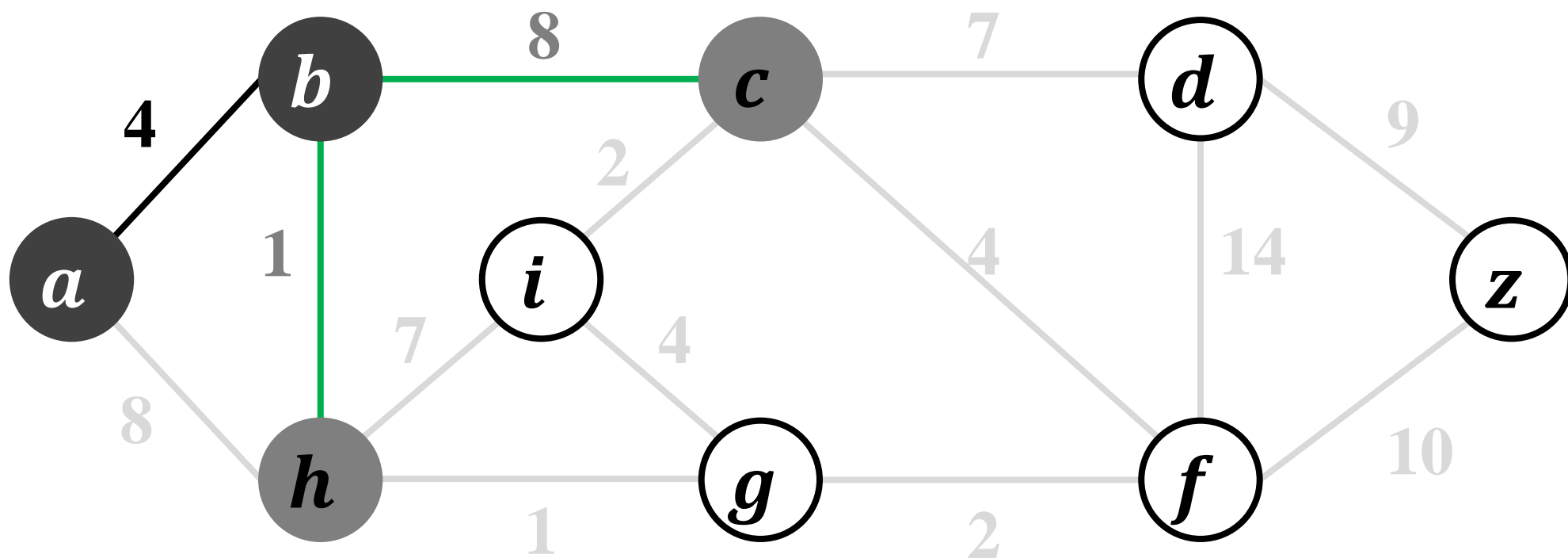
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	W	W	W	W	W	W	W
<i>dist</i>	0	4	∞	∞	∞	∞	8	∞	∞
<i>pred</i>	N	<i>a</i>	N	N	N	N	<i>a</i>	N	N



算法实例



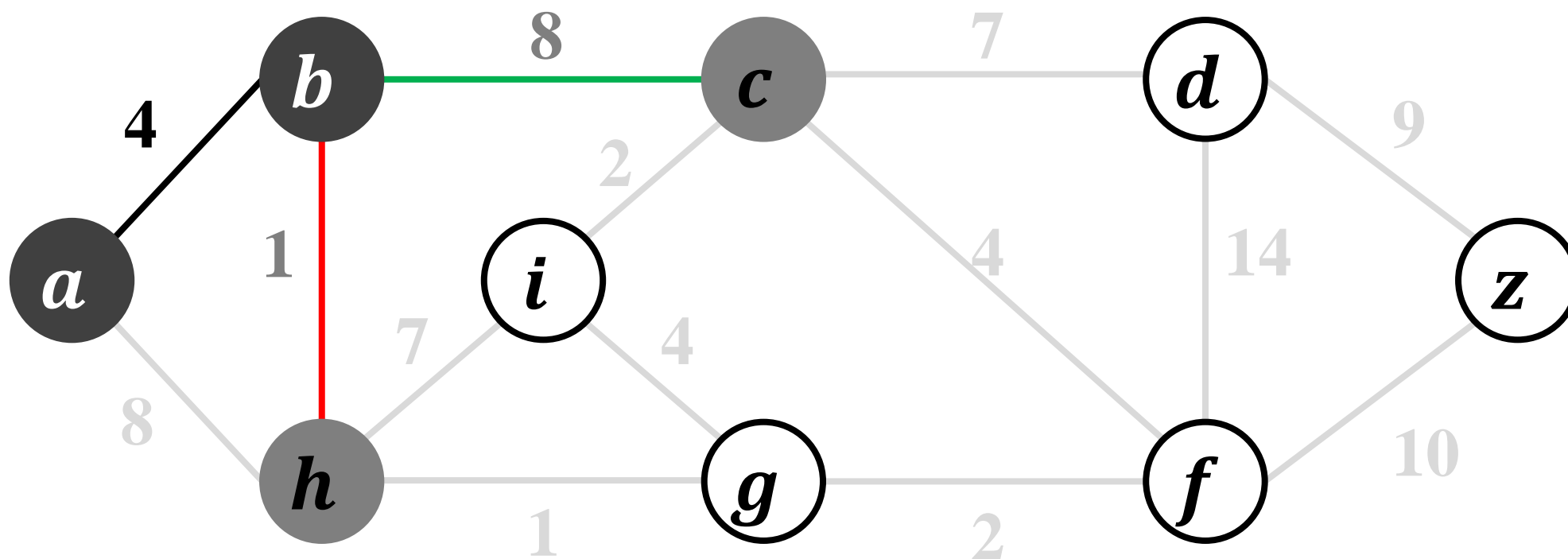
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	W	W	W	W	W	W	W
<i>dist</i>	0	4	8	∞	∞	∞	1	∞	∞
<i>pred</i>	N	<i>a</i>	<i>b</i>	N	N	N	<i>b</i>	N	N



算法实例



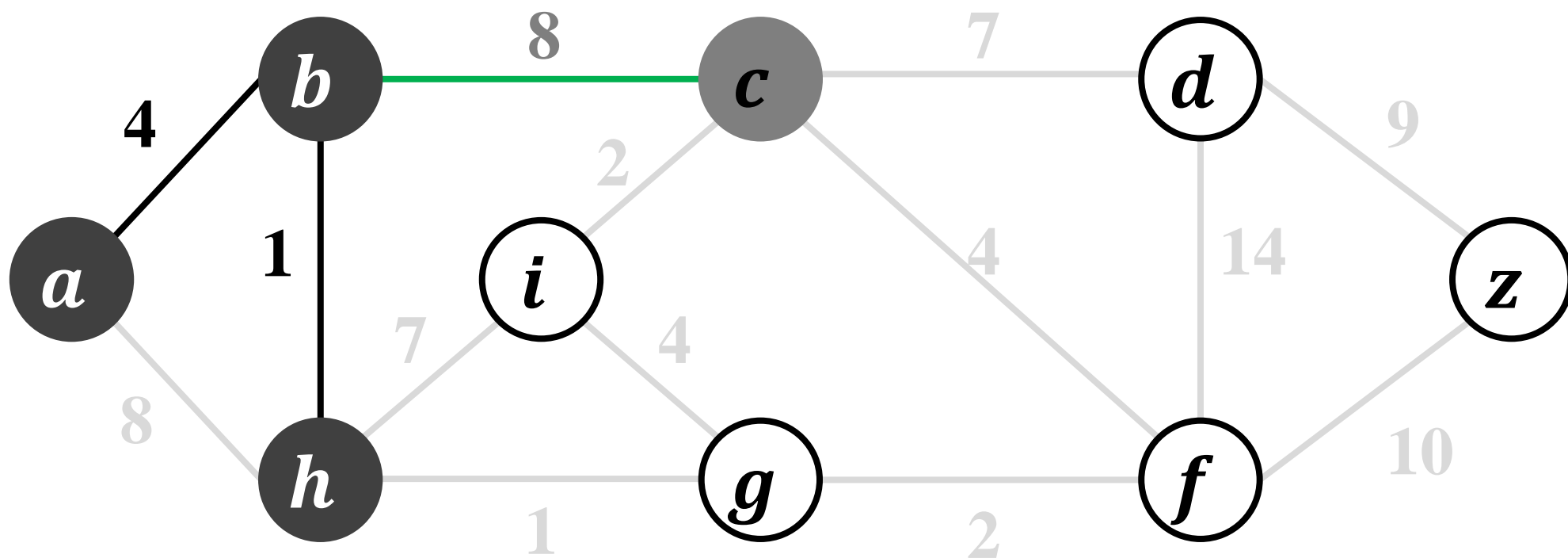
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	W	W	W	W	W	W	W
<i>dist</i>	0	4	8	∞	∞	∞	1	∞	∞
<i>pred</i>	N	<i>a</i>	<i>b</i>	N	N	N	<i>b</i>	N	N



算法实例



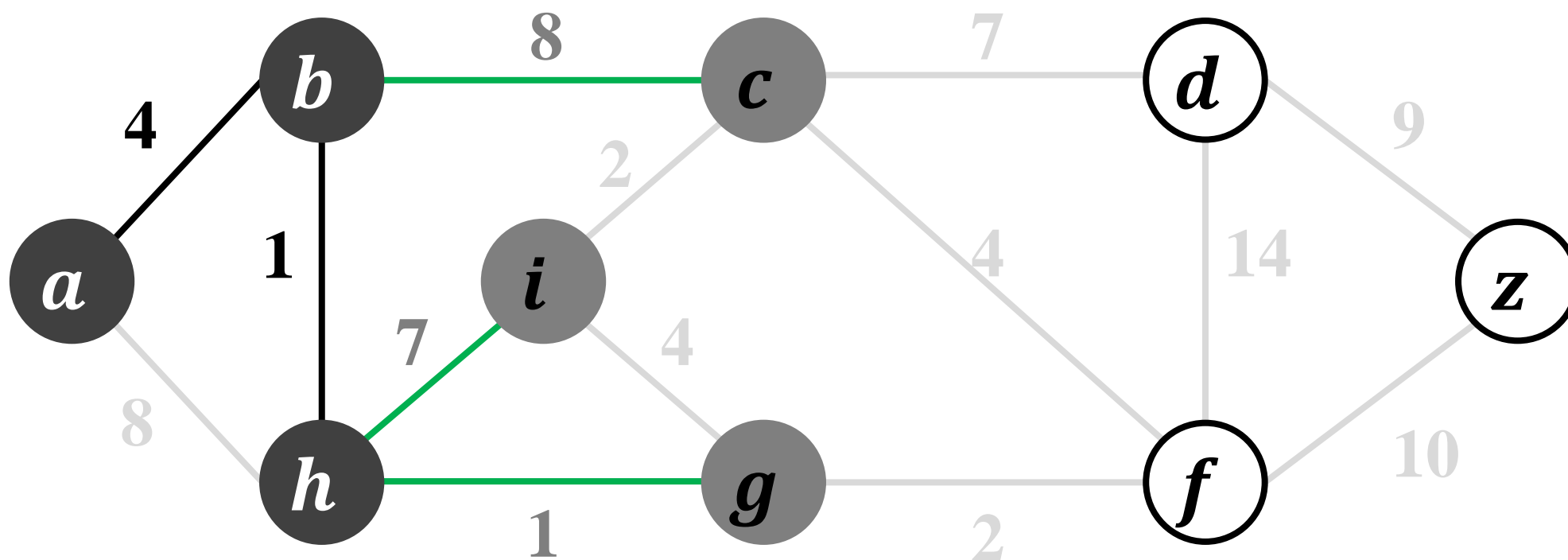
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	W	W	W	W	B	W	W
<i>dist</i>	0	4	8	∞	∞	∞	1	∞	∞
<i>pred</i>	N	<i>a</i>	<i>b</i>	N	N	N	<i>b</i>	N	N



算法实例



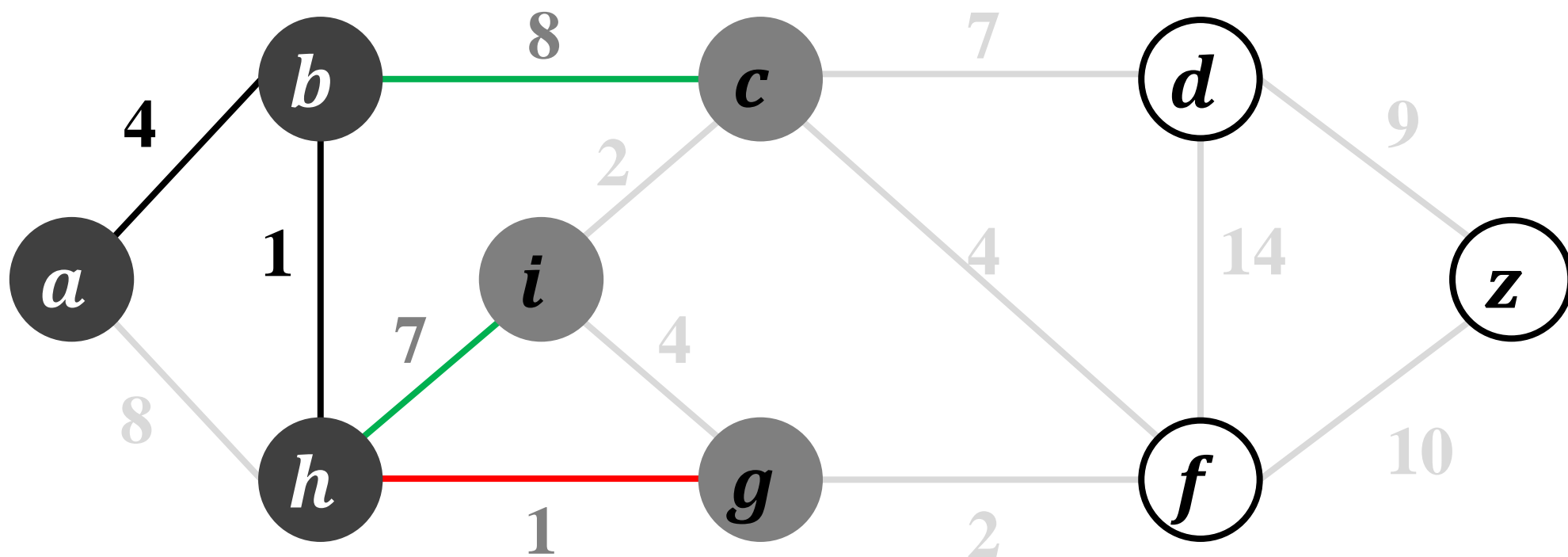
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	W	W	W	W	B	W	W
<i>dist</i>	0	4	8	∞	∞	1	1	7	∞
<i>pred</i>	N	<i>a</i>	<i>b</i>	N	N	<i>h</i>	<i>b</i>	<i>h</i>	N



算法实例



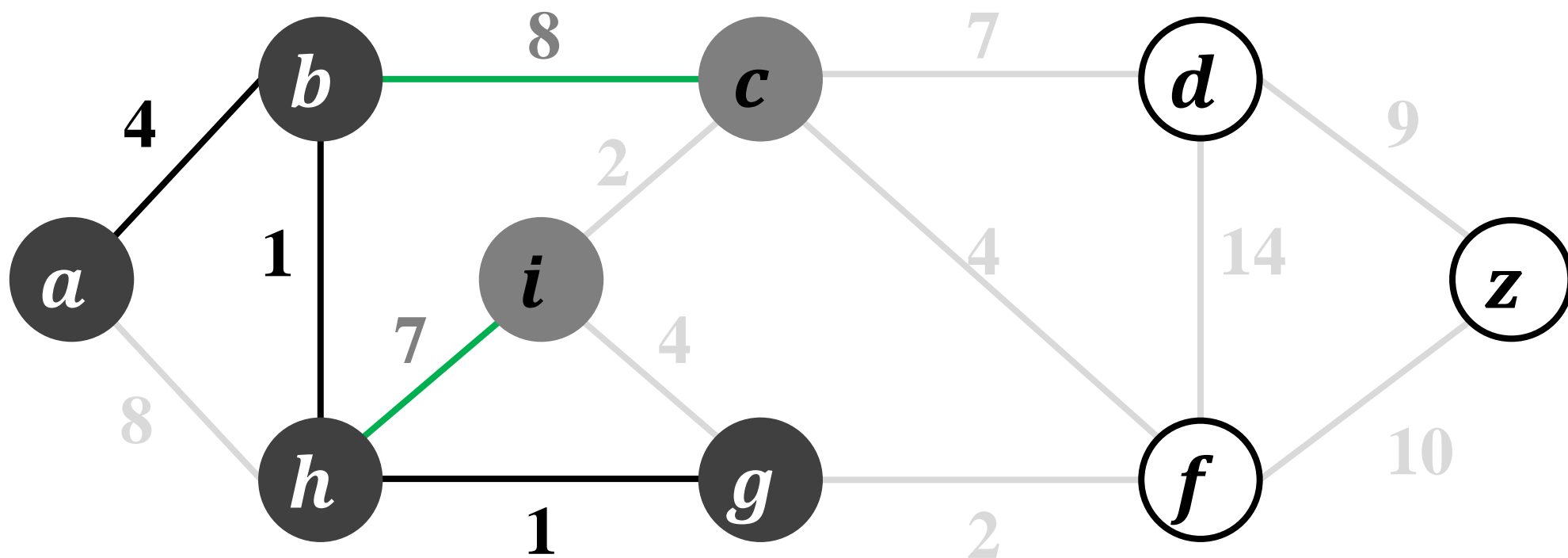
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	W	W	W	W	B	W	W
<i>dist</i>	0	4	8	∞	∞	1	1	7	∞
<i>pred</i>	N	<i>a</i>	<i>b</i>	N	N	<i>h</i>	<i>b</i>	<i>h</i>	N



算法实例



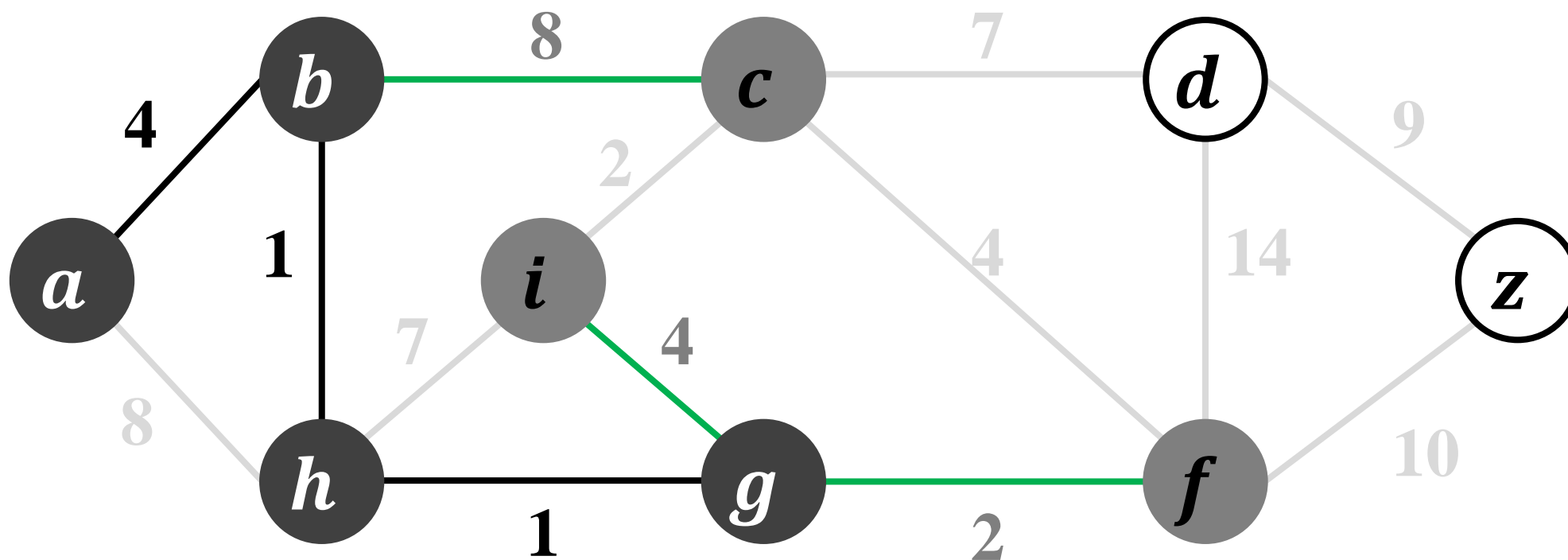
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	W	W	W	B	B	W	W
<i>dist</i>	0	4	8	∞	∞	1	1	7	∞
<i>pred</i>	N	<i>a</i>	<i>b</i>	N	N	<i>h</i>	<i>b</i>	<i>h</i>	N



算法实例



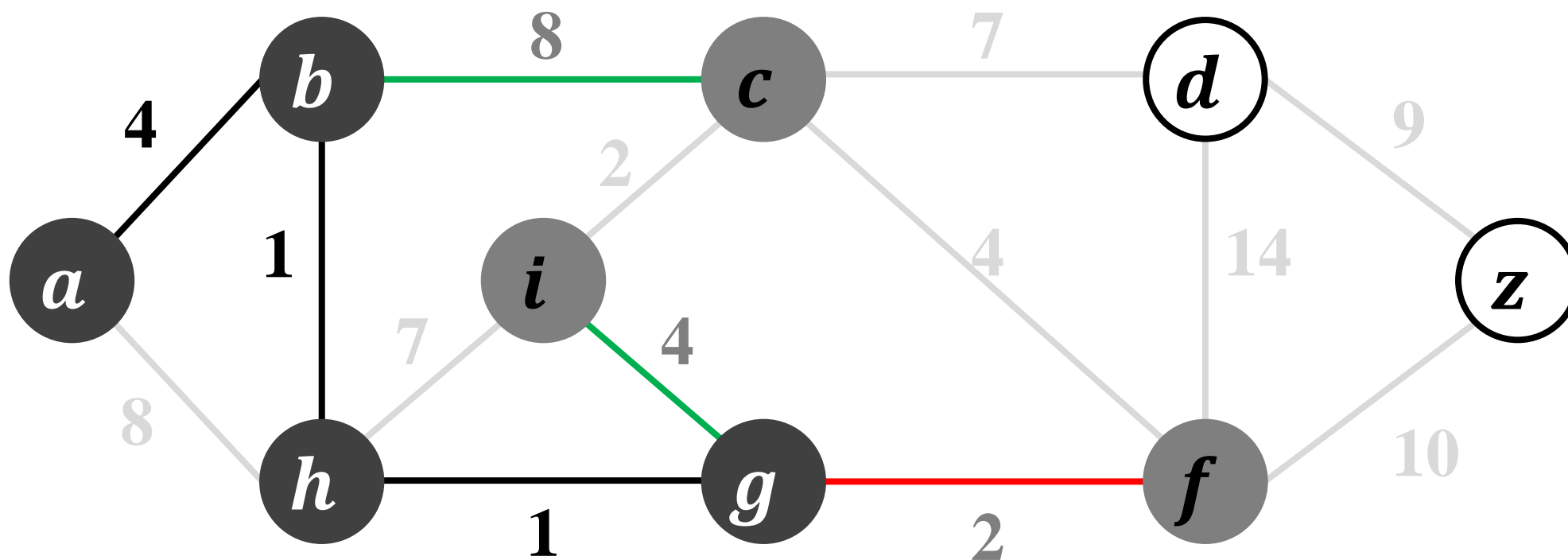
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	W	W	W	B	B	W	W
<i>dist</i>	0	4	8	∞	2	1	1	4	∞
<i>pred</i>	N	<i>a</i>	<i>b</i>	N	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	N



算法实例



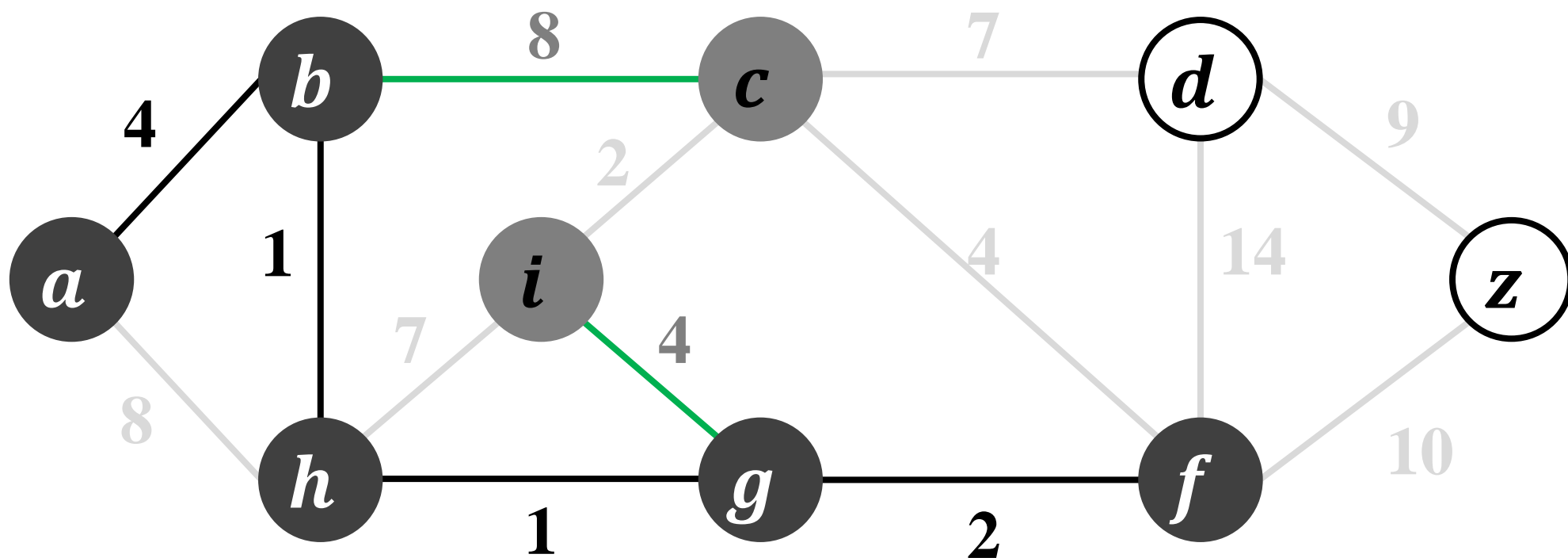
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	W	W	W	B	B	W	W
<i>dist</i>	0	4	8	∞	2	1	1	4	∞
<i>pred</i>	N	<i>a</i>	<i>b</i>	N	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	N



算法实例



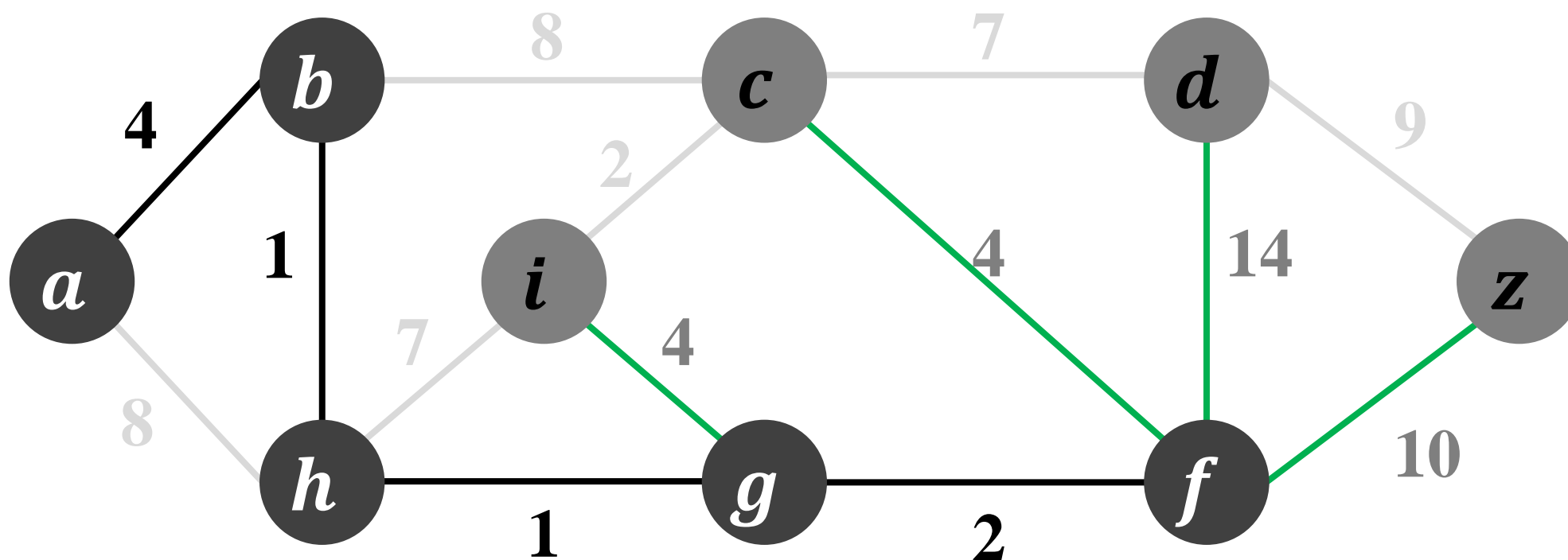
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	W	W	B	B	B	W	W
<i>dist</i>	0	4	8	∞	2	1	1	4	∞
<i>pred</i>	N	<i>a</i>	<i>b</i>	N	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	N



算法实例



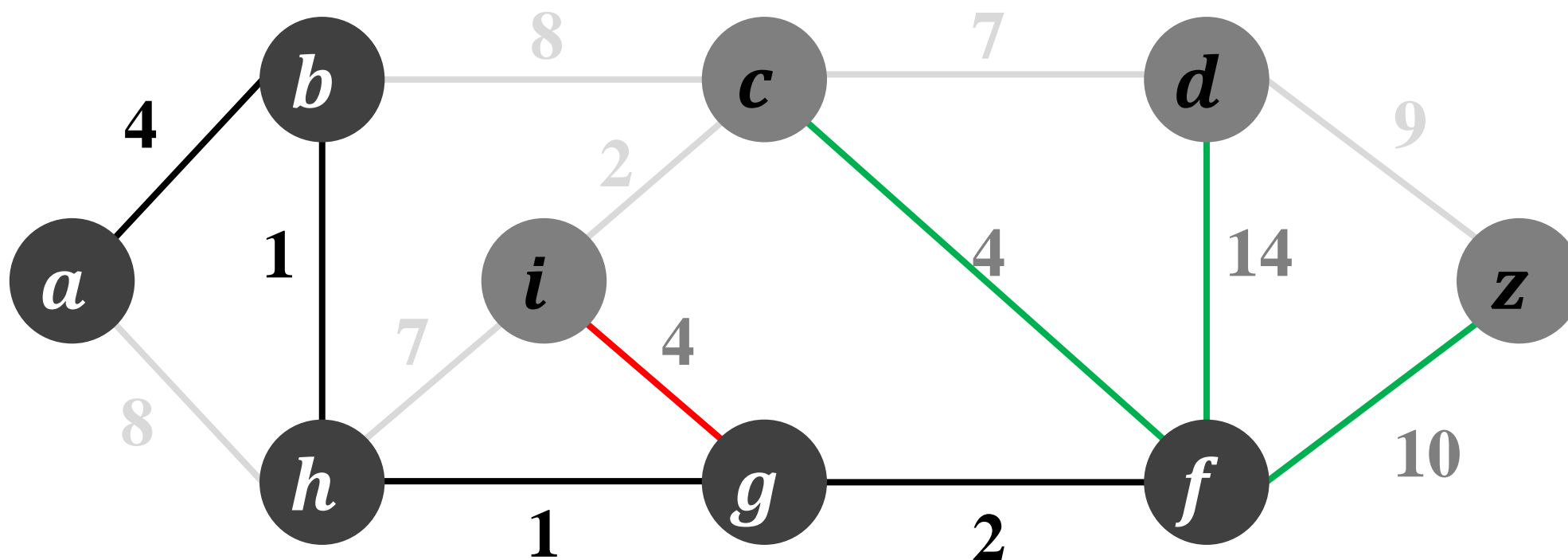
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	W	W	B	B	B	W	W
<i>dist</i>	0	4	4	14	2	1	1	4	10
<i>pred</i>	N	<i>a</i>	<i>f</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	<i>f</i>



算法实例



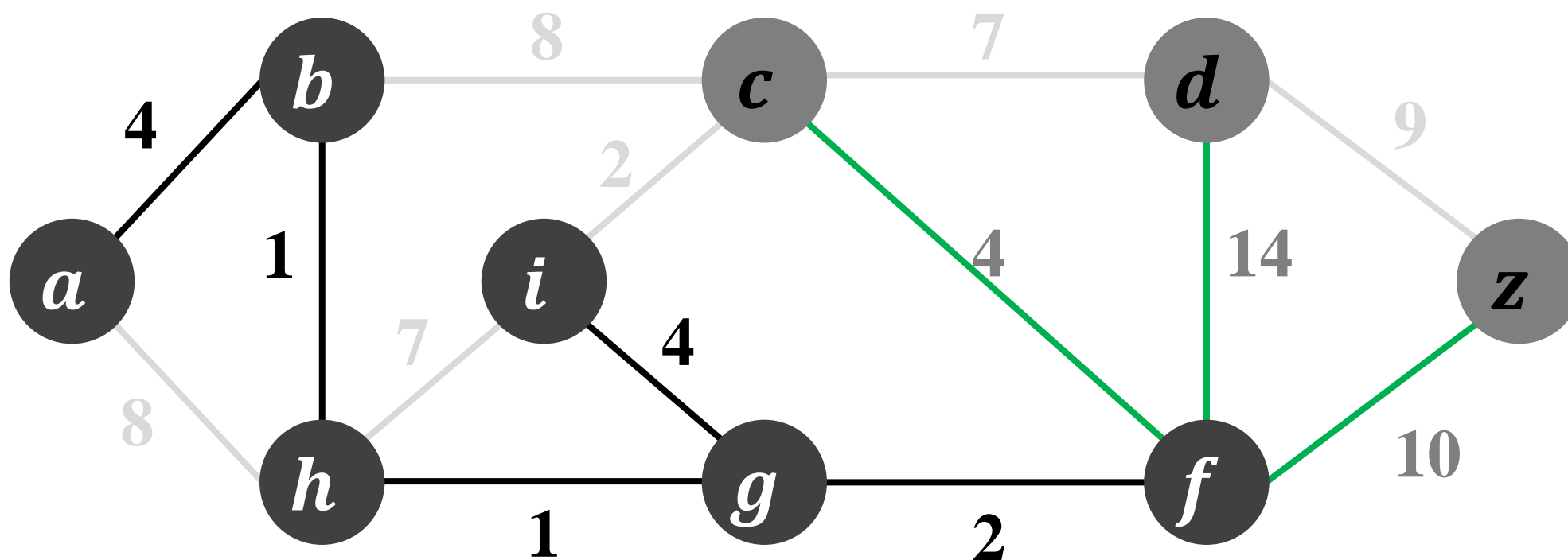
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	W	W	B	B	B	W	W
<i>dist</i>	0	4	4	14	2	1	1	4	10
<i>pred</i>	N	<i>a</i>	<i>f</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	<i>f</i>



算法实例



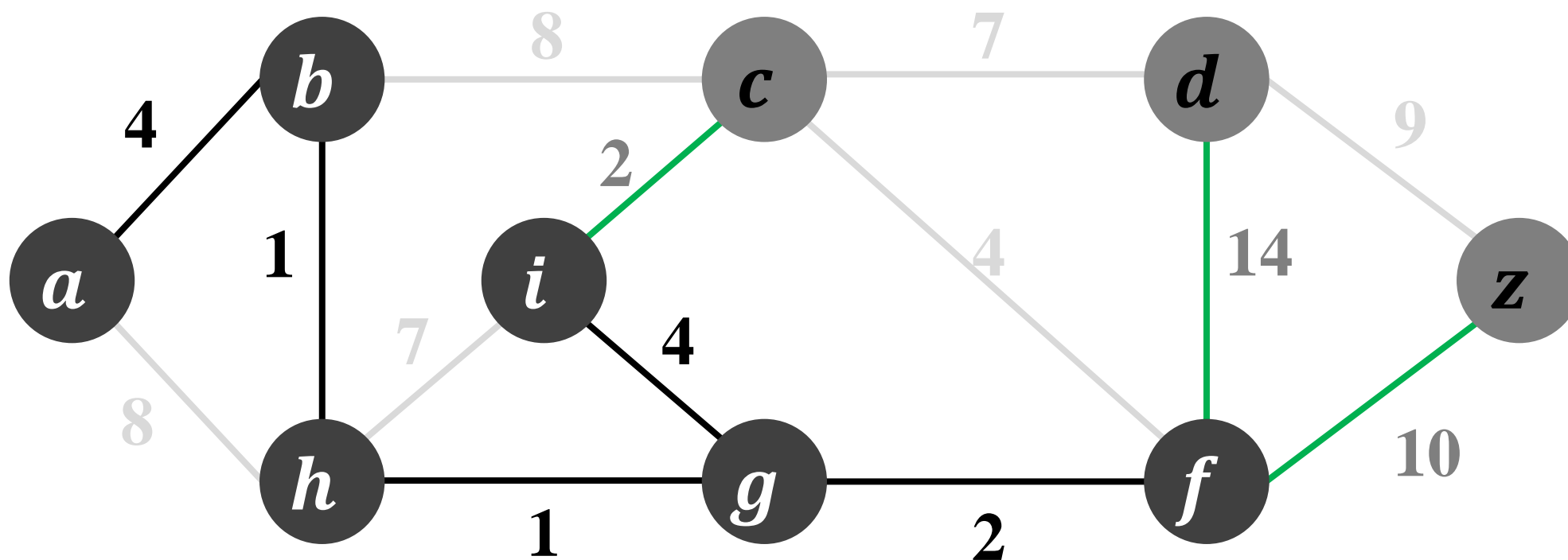
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	W	W	B	B	B	B	W
<i>dist</i>	0	4	4	14	2	1	1	4	10
<i>pred</i>	N	<i>a</i>	<i>f</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	<i>f</i>



算法实例



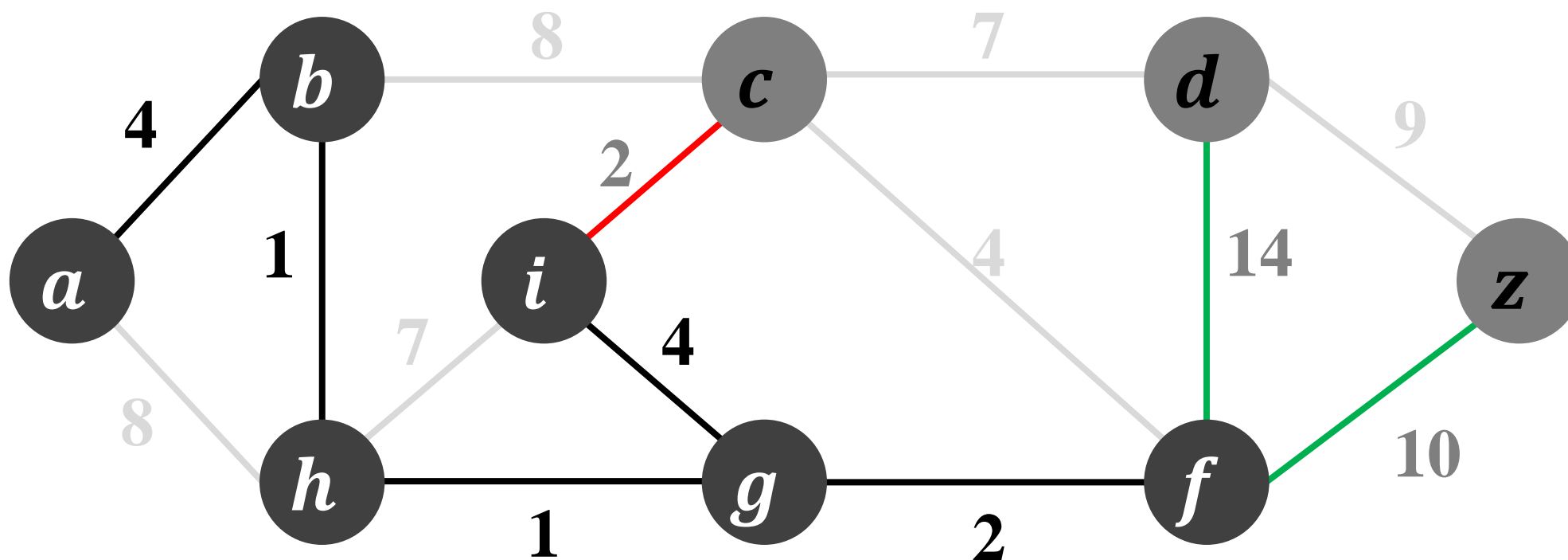
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	W	W	B	B	B	B	W
<i>dist</i>	0	4	2	14	2	1	1	4	10
<i>pred</i>	N	<i>a</i>	<i>i</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	<i>f</i>



算法实例



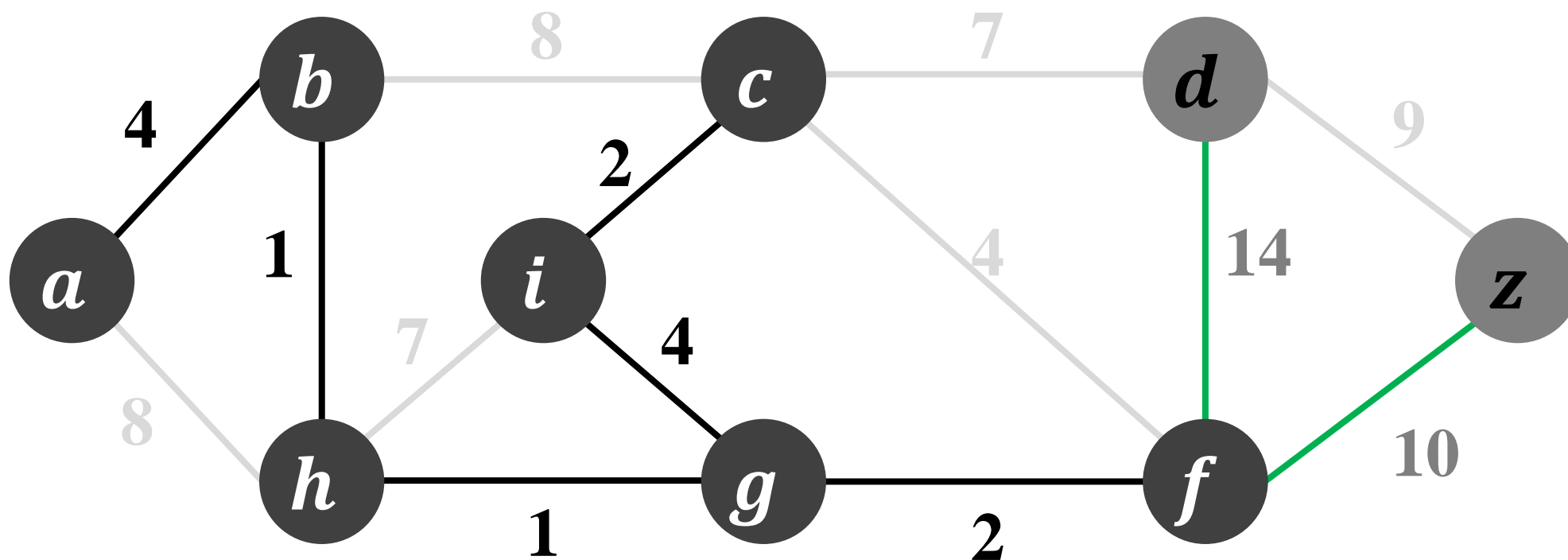
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	W	W	B	B	B	B	W
<i>dist</i>	0	4	2	14	2	1	1	4	10
<i>pred</i>	N	<i>a</i>	<i>i</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	<i>f</i>



算法实例



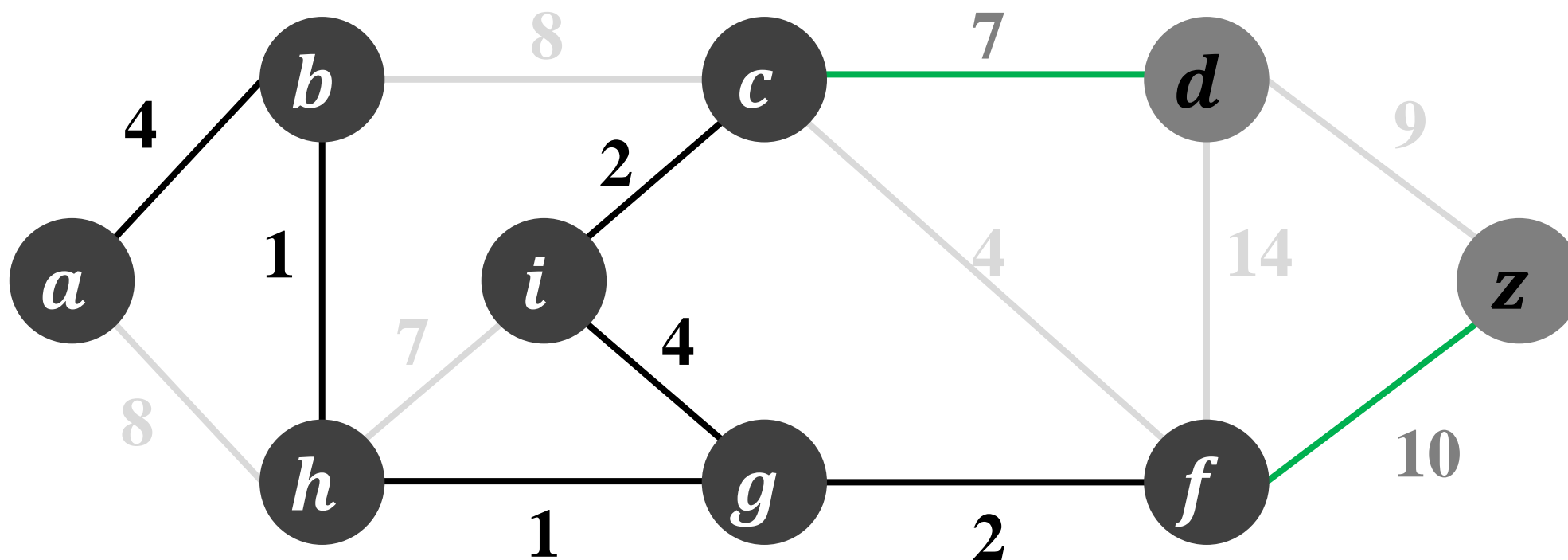
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	B	W	B	B	B	B	W
<i>dist</i>	0	4	2	14	2	1	1	4	10
<i>pred</i>	N	<i>a</i>	<i>i</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	<i>f</i>



算法实例



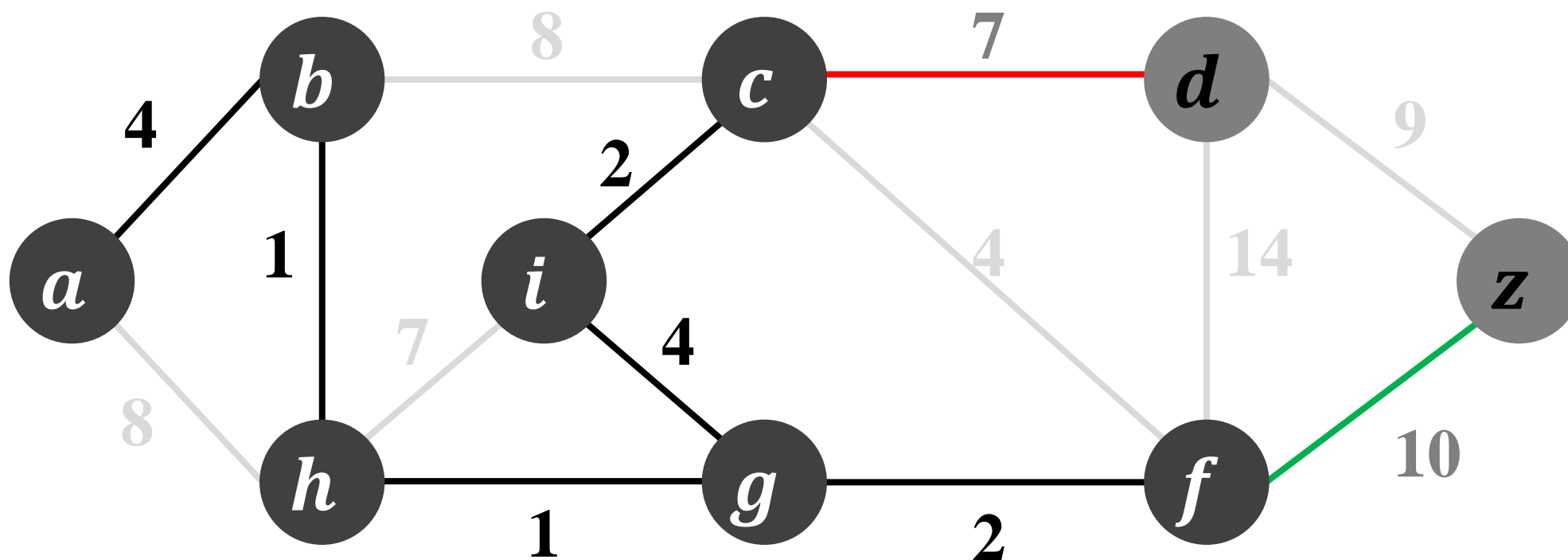
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	B	W	B	B	B	B	W
<i>dist</i>	0	4	2	7	2	1	1	4	10
<i>pred</i>	N	<i>a</i>	<i>i</i>	<i>c</i>	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	<i>f</i>



算法实例



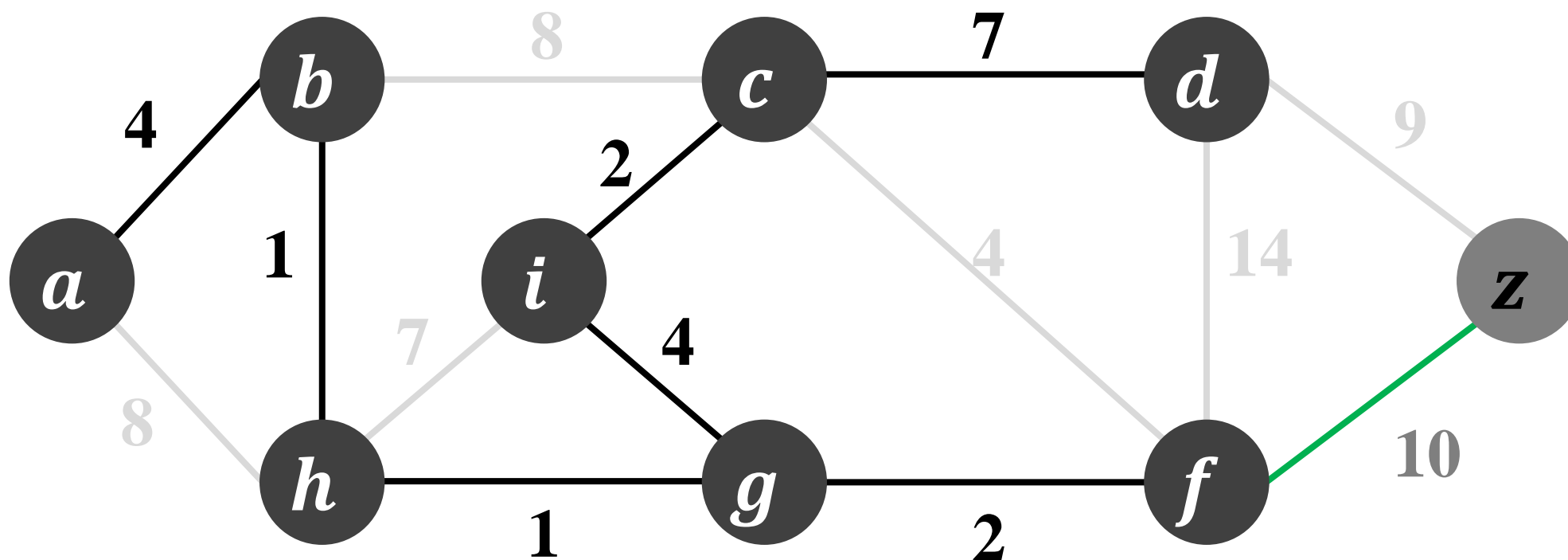
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	B	W	B	B	B	B	W
<i>dist</i>	0	4	2	7	2	1	1	4	10
<i>pred</i>	N	<i>a</i>	<i>i</i>	<i>c</i>	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	<i>f</i>



算法实例



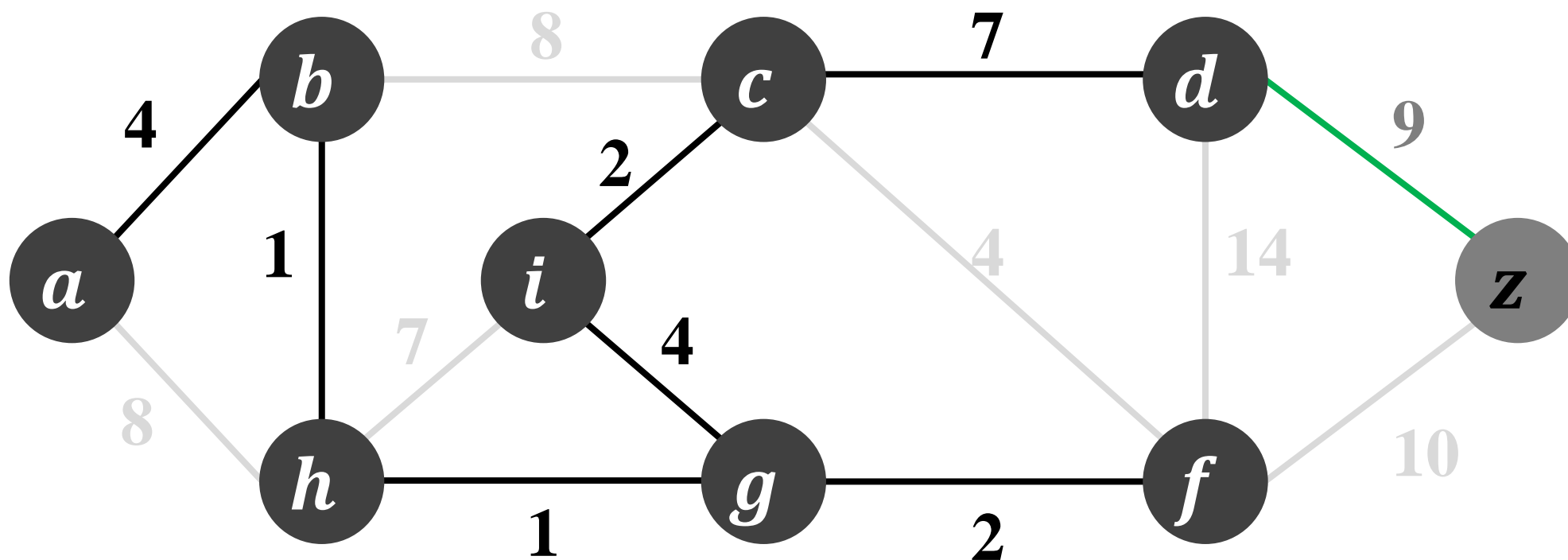
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	B	B	B	B	B	B	W
<i>dist</i>	0	4	2	7	2	1	1	4	10
<i>pred</i>	N	<i>a</i>	<i>i</i>	<i>c</i>	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	<i>f</i>



算法实例



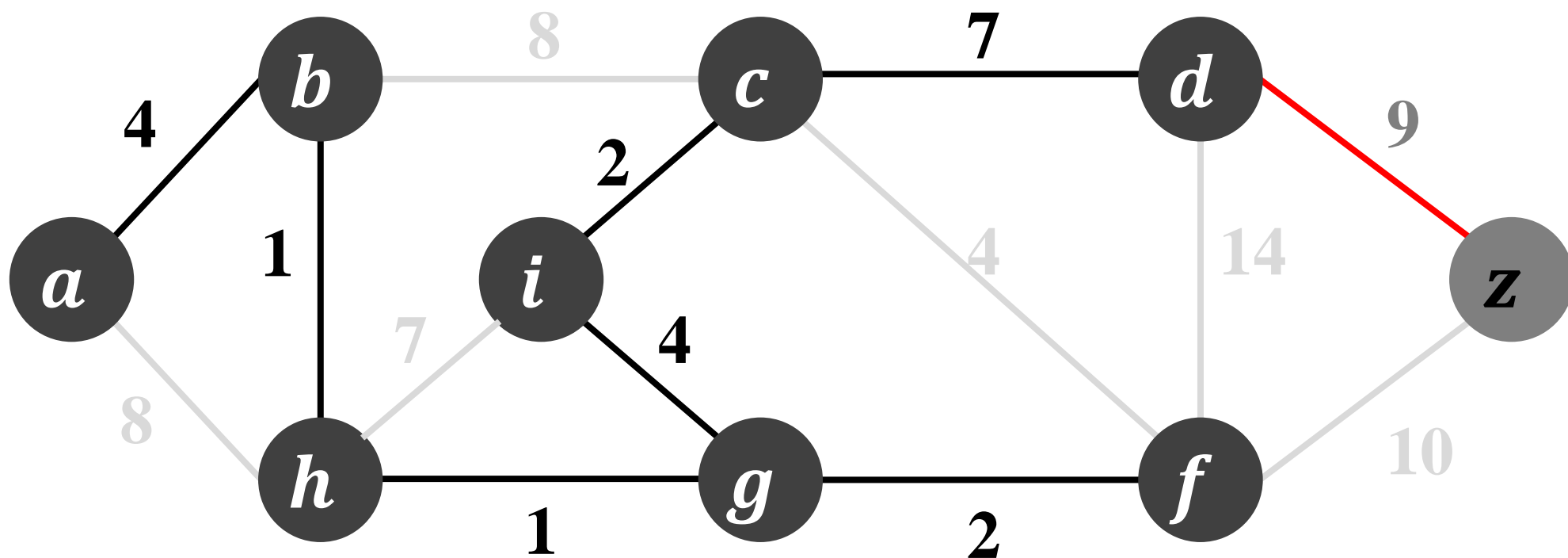
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	B	B	B	B	B	B	W
<i>dist</i>	0	4	2	7	2	1	1	4	9
<i>pred</i>	N	<i>a</i>	<i>i</i>	<i>c</i>	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	<i>d</i>



算法实例



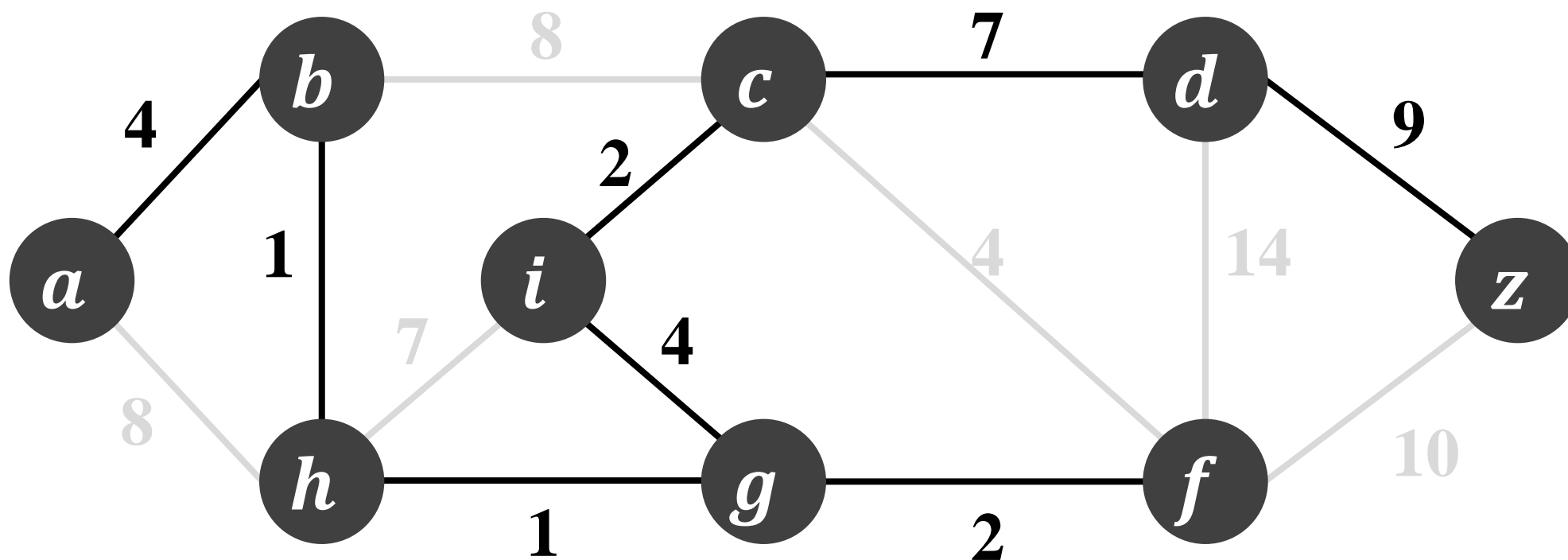
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	B	B	B	B	B	B	W
<i>dist</i>	0	4	2	7	2	1	1	4	9
<i>pred</i>	N	<i>a</i>	<i>i</i>	<i>c</i>	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	<i>d</i>



算法实例



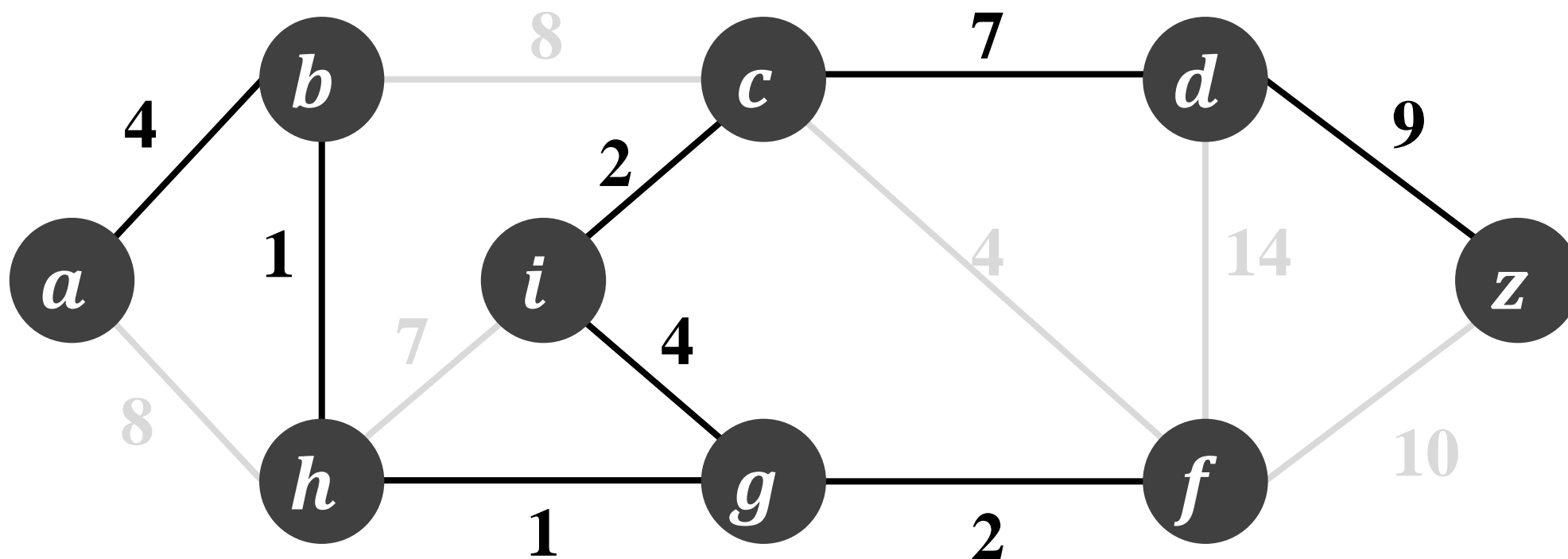
<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	B	B	B	B	B	B	B
<i>dist</i>	0	4	2	7	2	1	1	4	9
<i>pred</i>	N	<i>a</i>	<i>i</i>	<i>c</i>	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	<i>d</i>



算法实例



<i>V</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>z</i>
<i>color</i>	B	B	B	B	B	B	B	B	B
<i>dist</i>	0	4	2	7	2	1	1	4	9
<i>pred</i>	N	<i>a</i>	<i>i</i>	<i>c</i>	<i>g</i>	<i>h</i>	<i>b</i>	<i>g</i>	<i>d</i>



$$W(T) = 0 + 4 + 2 + 7 + 2 + 1 + 1 + 4 + 9 = 30$$

问题背景

通用框架

Prim算法

算法实例

算法分析

- **MST-Prim(G)**

输入: 图 $G = \langle V, E, W \rangle$

输出: 最小生成树 T

新建一维数组 $color[1..|V|]$, $dist[1..|V|]$, $pred[1..|V|]$

//初始化

```
for  $u \in V$  do
     $color[u] \leftarrow WHITE$ 
     $dist[u] \leftarrow \infty$ 
     $pred[u] \leftarrow NULL$ 
end
 $dist[1] \leftarrow 0$ 
```

初始化各个辅助数组

- **MST-Prim(G)**

输入: 图 $G = \langle V, E, W \rangle$

输出: 最小生成树 T

新建一维数组 $color[1..|V|]$, $dist[1..|V|]$, $pred[1..|V|]$

//初始化

for $u \in V$ do

$color[u] \leftarrow WHITE$

$dist[u] \leftarrow \infty$

$pred[u] \leftarrow NULL$

end

$dist[1] \leftarrow 0$

选择任意顶点作为起点

● MST-Prim(G)

~~//执行最小生成树算法~~

~~for~~ $i \leftarrow 1$ to $|V|$ ~~do~~

~~$minDist \leftarrow \infty$~~

$rec \leftarrow 0$

for $j \leftarrow 1$ to $|V|$ do

 if $color[j] \neq BLACK$ and $dist[j] < minDist$ then

$minDist \leftarrow dist[j]$

$rec \leftarrow j$

 end

end

for $u \in G.Adj[rec]$ do

 if $w(rec, u) < dist[u]$ then

$dist[u] \leftarrow w(rec, u)$

$pred[u] \leftarrow rec$

 end

end

$color[rec] \leftarrow BLACK$

end

依次添加其他顶点

● MST-Prim(G)

//执行最小生成树算法

```
for  $i \leftarrow 1$  to  $|V|$  do
     $minDist \leftarrow \infty$ 
     $rec \leftarrow 0$ 
    for  $j \leftarrow 1$  to  $|V|$  do
        if  $color[j] \neq BLACK$  and  $dist[j] < minDist$  then
             $minDist \leftarrow dist[j]$ 
             $rec \leftarrow j$ 
        end
    end
    for  $u \in G.Adj[rec]$  do
        if  $w(rec, u) < dist[u]$  then
             $dist[u] \leftarrow w(rec, u)$ 
             $pred[u] \leftarrow rec$ 
        end
    end
     $color[rec] \leftarrow BLACK$ 
end
```

记录最小权值

● MST-Prim(G)

//执行最小生成树算法

for $i \leftarrow 1$ to $|V|$ do

~~$minDist \leftarrow \infty$~~

$rec \leftarrow 0$

~~for $j \leftarrow 1$ to $|V|$ do~~

~~if $color[j] \neq BLACK$ and $dist[j] < minDist$ then~~

~~$minDist \leftarrow dist[j]$~~

~~$rec \leftarrow j$~~

~~end~~

end

for $u \in G.Adj[rec]$ do

 if $w(rec, u) < dist[u]$ then

$dist[u] \leftarrow w(rec, u)$

$pred[u] \leftarrow rec$

 end

end

$color[rec] \leftarrow BLACK$

end

记录安全边的端点

● MST-Prim(G)

//执行最小生成树算法

for $i \leftarrow 1$ to $|V|$ do

$minDist \leftarrow \infty$

$rec \leftarrow 0$

 for $j \leftarrow 1$ to $|V|$ do

 if $color[j] \neq BLACK$ and $dist[j] < minDist$ then

$minDist \leftarrow dist[j]$

$rec \leftarrow j$

 end

 end

 for $u \in G.Adj[rec]$ do

 if $w(rec, u) < dist[u]$ then

$dist[u] \leftarrow w(rec, u)$

$pred[u] \leftarrow rec$

 end

 end

$color[rec] \leftarrow BLACK$

end

记录新增的安全边

● MST-Prim(G)

//执行最小生成树算法

```
for  $i \leftarrow 1$  to  $|V|$  do
   $minDist \leftarrow \infty$ 
   $rec \leftarrow 0$ 
  for  $j \leftarrow 1$  to  $|V|$  do
    if  $color[j] \neq BLACK$  and  $dist[j] < minDist$  then
       $minDist \leftarrow dist[j]$ 
       $rec \leftarrow j$ 
    end
  end
  for  $u \in G.Adj[rec]$  do
    if  $w(rec, u) < dist[u]$  then
       $dist[u] \leftarrow w(rec, u)$ 
       $pred[u] \leftarrow rec$ 
    end
  end
   $color[rec] \leftarrow BLACK$ 
end
```

更新 $dist$ 数组

- **MST-Prim(G)**

//执行最小生成树算法

```
for  $i \leftarrow 1$  to  $|V|$  do
   $minDist \leftarrow \infty$ 
   $rec \leftarrow 0$ 
  for  $j \leftarrow 1$  to  $|V|$  do
    if  $color[j] \neq BLACK$  and  $dist[j] < minDist$  then
       $minDist \leftarrow dist[j]$ 
       $rec \leftarrow j$ 
    end
  end
  for  $u \in G.Adj[rec]$  do
    if  $w(rec, u) < dist[u]$  then
       $dist[u] \leftarrow w(rec, u)$ 
       $pred[u] \leftarrow rec$ 
    end
  end
end
 $color[rec] \leftarrow BLACK$ 
end
```

标记顶点处理完成

- MST-Prim(G)

输入: 图 $G = \langle V, E, W \rangle$

输出: 最小生成树 T

新建一维数组 $color[1..|V|]$, $dist[1..|V|]$, $pred[1..|V|]$

//初始化

for $u \in V$ do

$color[u] \leftarrow WHITE$

$dist[u] \leftarrow \infty$

$pred[u] \leftarrow NULL$

end

$dist[1] \leftarrow 0$

} $O(|V|)$

- **MST-Prim(G)**

//执行最小生成树算法

```
for  $i \leftarrow 1$  to  $|V|$  do
```

```
     $minDist \leftarrow \infty$ 
```

```
     $rec \leftarrow 0$ 
```

```
    for  $j \leftarrow 1$  to  $|V|$  do
```

```
        if  $color[j] \neq BLACK$  and  $dist[j] < minDist$  then
```

```
             $minDist \leftarrow dist[j]$ 
```

```
             $rec \leftarrow j$ 
```

```
        end
```

```
    end
```

```
    for  $u \in G.Adj[rec]$  do
```

```
        if  $w(rec, u) < dist[u]$  then
```

```
             $dist[u] \leftarrow w(rec, u)$ 
```

```
             $pred[u] \leftarrow rec$ 
```

```
        end
```

```
    end
```

```
     $color[rec] \leftarrow BLACK$ 
```

```
end
```

$O(|V|)$

$O(\deg(u))$

- MST-Prim(G)

//执行最小生成树算法

```
for  $i \leftarrow 1$  to  $|V|$  do
```

```
     $minDist \leftarrow \infty$ 
```

```
     $rec \leftarrow 0$ 
```

```
    for  $j \leftarrow 1$  to  $|V|$  do
```

```
        if  $color[j] \neq BLACK$  and  $dist[j] < minDist$  then
```

```
             $minDist \leftarrow dist[j]$ 
```

```
             $rec \leftarrow j$ 
```

```
        end
```

```
    end
```

```
    for  $u \in G.Adj[rec]$  do
```

```
        if  $w(rec, u) < dist[u]$  then
```

```
             $dist[u] \leftarrow w(rec, u)$ 
```

```
             $pred[u] \leftarrow rec$ 
```

```
        end
```

```
    end
```

```
     $color[rec] \leftarrow BLACK$ 
```

```
end
```

$O(|V|)$

$$O(|V| \cdot |V|) = O(|V|^2)$$

$O(\deg(u))$

- MST-Prim(G)

//执行最小生成树算法

```
for  $i \leftarrow 1$  to  $|V|$  do
   $minDist \leftarrow \infty$ 
   $rec \leftarrow 0$ 
  for  $j \leftarrow 1$  to  $|V|$  do
    if  $color[j] \neq BLACK$  and  $dist[j] < minDist$  then
       $minDist \leftarrow dist[j]$ 
       $rec \leftarrow j$ 
    end
  end
  for  $u \in G.Adj[rec]$  do
    if  $w(rec, u) < dist[u]$  then
       $dist[u] \leftarrow w(rec, u)$ 
       $pred[u] \leftarrow rec$ 
    end
  end
   $color[rec] \leftarrow BLACK$ 
end
```

$O(|V|)$

$$O(|V| \cdot |V|) = O(|V|^2)$$

$O(\deg(u))$

$$\sum_{u \in V} \deg(u) = 2|E|$$

- **MST-Prim(G)**

//执行最小生成树算法

```
for  $i \leftarrow 1$  to  $|V|$  do
```

```
     $minDist \leftarrow \infty$ 
```

```
     $rec \leftarrow 0$ 
```

```
    for  $j \leftarrow 1$  to  $|V|$  do
```

```
        if  $color[j] \neq BLACK$  and  $dist[j] < minDist$  then
```

```
             $minDist \leftarrow dist[j]$ 
```

```
             $rec \leftarrow j$ 
```

```
        end
```

```
    end
```

```
    for  $u \in G.Adj[rec]$  do
```

```
        if  $w(rec, u) < dist[u]$  then
```

```
             $dist[u] \leftarrow w(rec, u)$ 
```

```
             $pred[u] \leftarrow rec$ 
```

```
        end
```

```
    end
```

```
     $color[rec] \leftarrow BLACK$ 
```

```
end
```

$O(|V|)$

$O(\deg(u))$

$O(|V|^2 + |E|)$

$O(|V|^2)$

● MST-Prim(G)

//执行最小生成树算法

```
for  $i \leftarrow 1$  to  $|V|$  do
   $minDist \leftarrow \infty$ 
   $rec \leftarrow 0$ 
  for  $j \leftarrow 1$  to  $|V|$  do
    if  $color[j] \neq BLACK$  and  $dist[j] < minDist$  then
       $minDist \leftarrow dist[j]$ 
       $rec \leftarrow j$ 
    end
  end
  for  $u \in G.Adj[rec]$  do
    if  $w(rec, u) < dist[u]$  then
       $dist[u] \leftarrow w(rec, u)$ 
       $pred[u] \leftarrow rec$ 
    end
  end
   $color[rec] \leftarrow BLACK$ 
end
```

$O(|V|)$

$O(|V|^2 + |E|)$

$O(\deg(u))$


$O(|V|^2)$

问题：能否进一步优化？

● MST-Prim(G)

//执行最小生成树算法

```
for  $i \leftarrow 1$  to  $|V|$  do
   $minDist \leftarrow \infty$ 
   $rec \leftarrow 0$ 
  for  $j \leftarrow 1$  to  $|V|$  do
    if  $color[j] \neq BLACK$  and  $dist[j] < minDist$  then
       $minDist \leftarrow dist[j]$ 
       $rec \leftarrow j$ 
    end
  end
  for  $u \in G.Adj[rec]$  do
    if  $w(rec, u) < dist[u]$  then
       $dist[u] \leftarrow w(rec, u)$ 
       $pred[u] \leftarrow rec$ 
    end
  end
   $color[rec] \leftarrow BLACK$ 
end
```

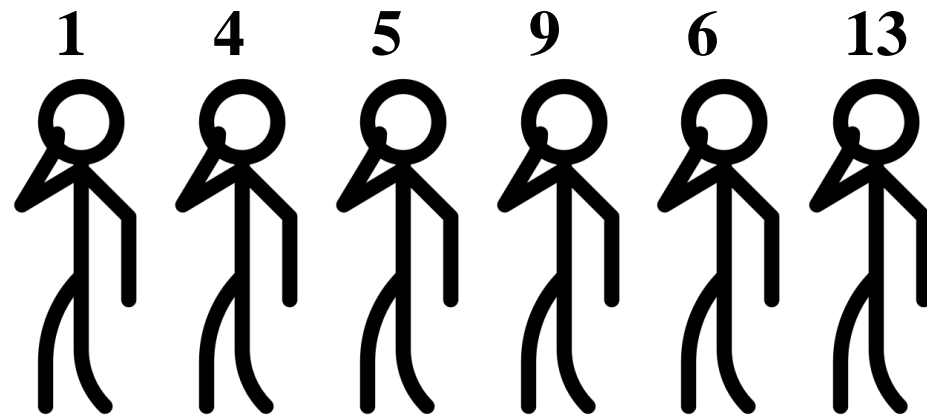
$O(|V|)$  $O(?)$

$O(|V|^2)$

数据结构加速最小值查询

优先队列

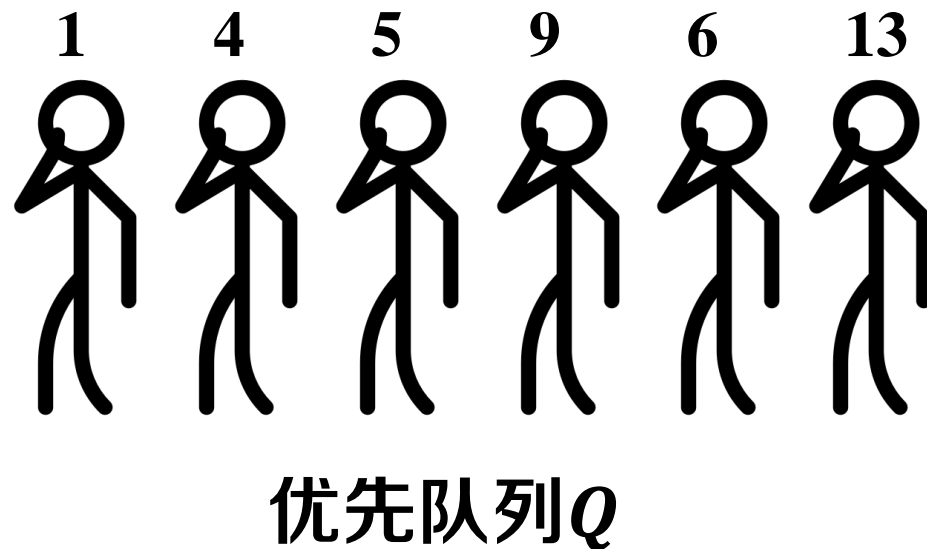
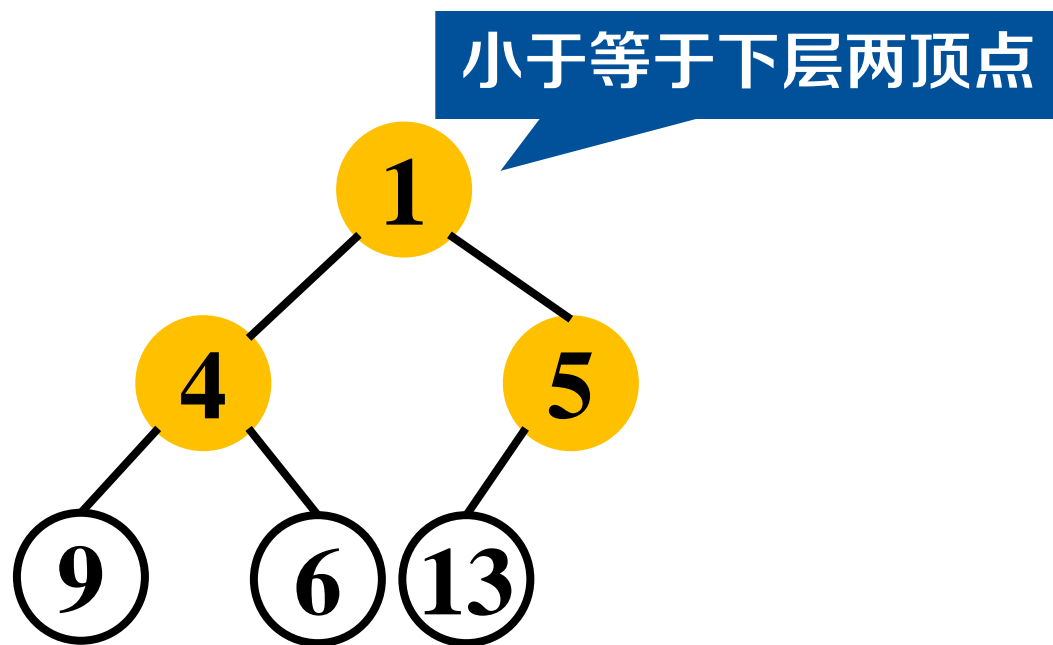
- 队列中每个元素有一个关键字，依据**关键字大小**离开队列



优先队列 Q

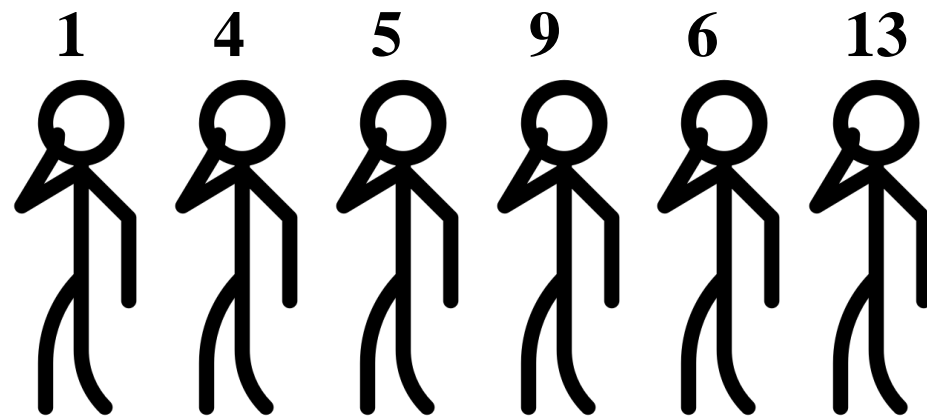
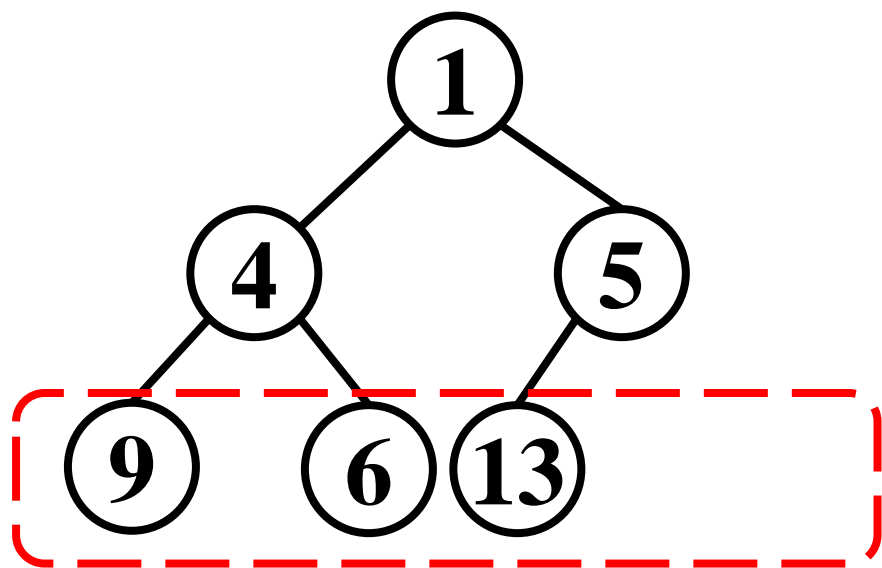
优先队列

- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列



优先队列

- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列

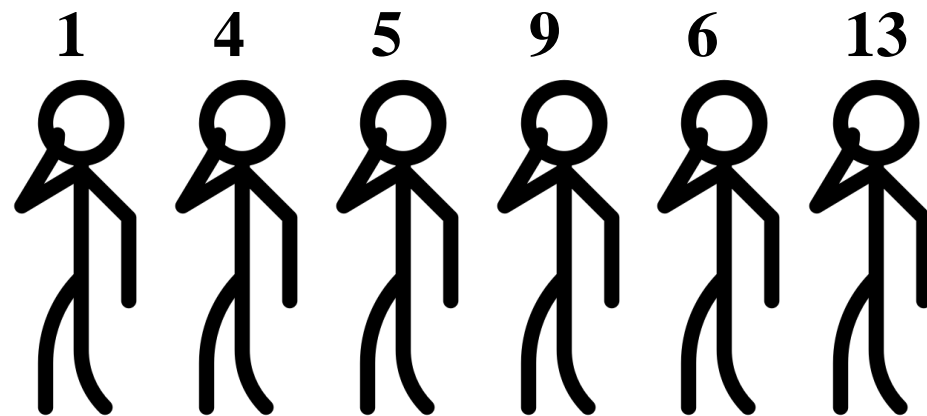
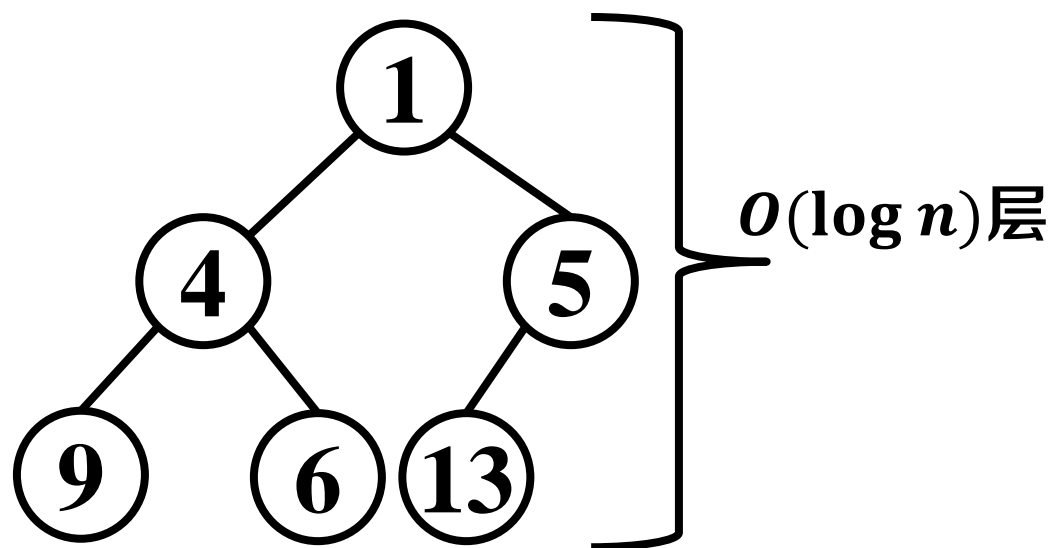


优先队列 Q

除最底层，第 h 层有 2^{h-1} 个顶点

优先队列

- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列

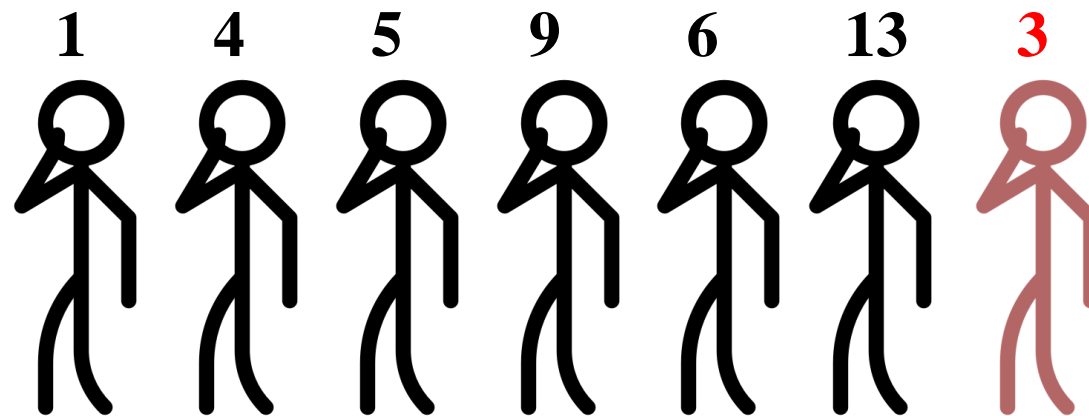
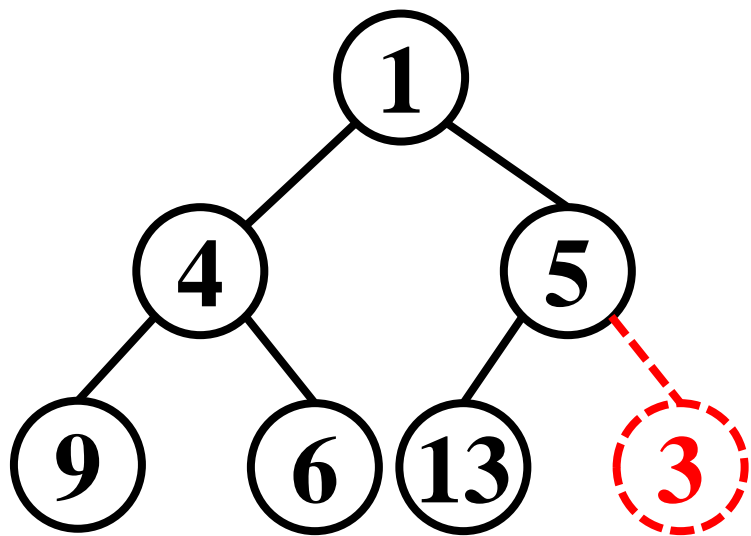


优先队列 Q

除最底层，第 h 层有 2^{h-1} 个顶点

优先队列

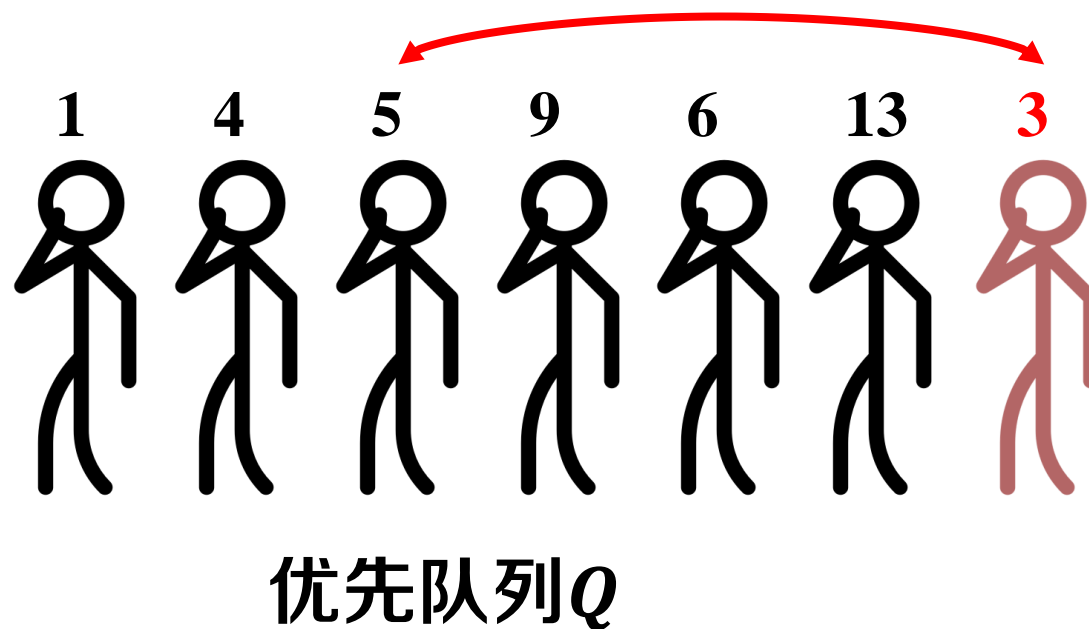
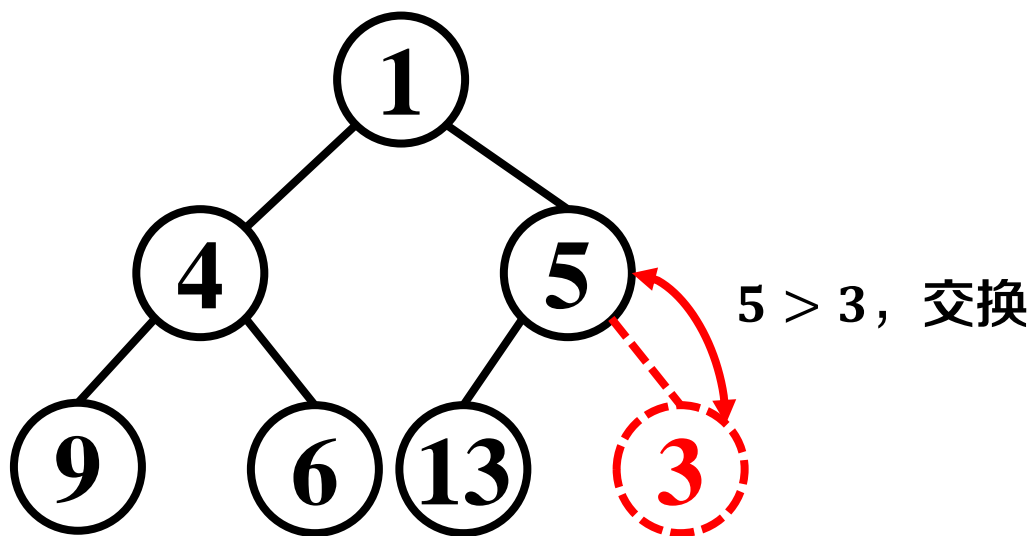
- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - *Q.Insert()*



优先队列 Q

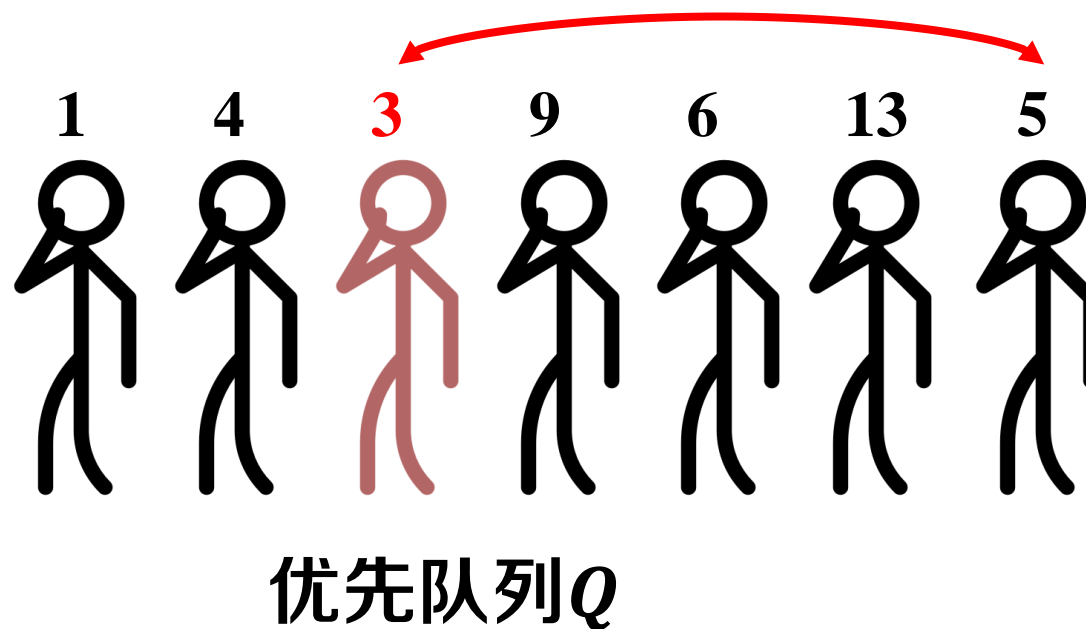
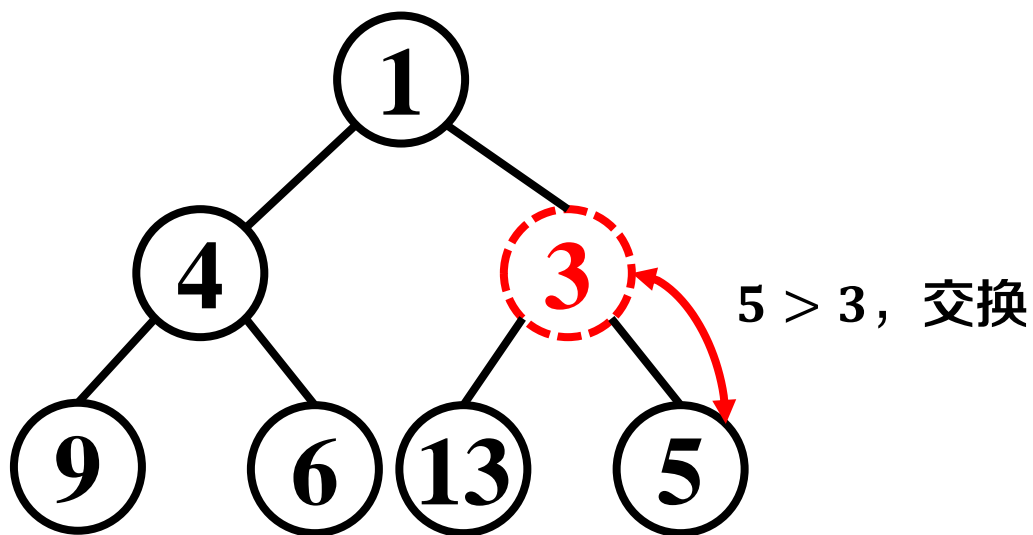
优先队列

- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - $Q.Insert()$



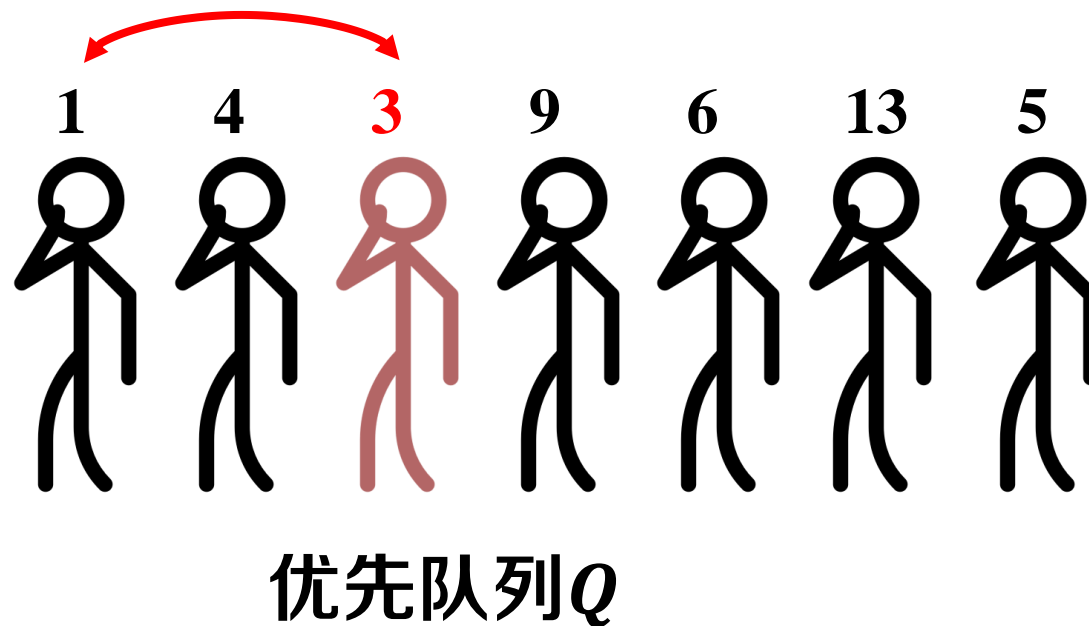
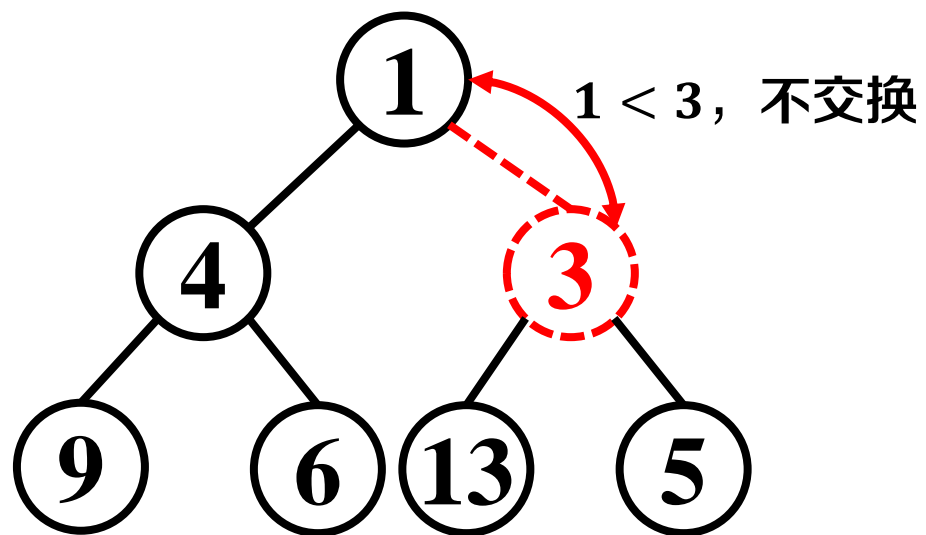
优先队列

- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - *Q.Insert()*



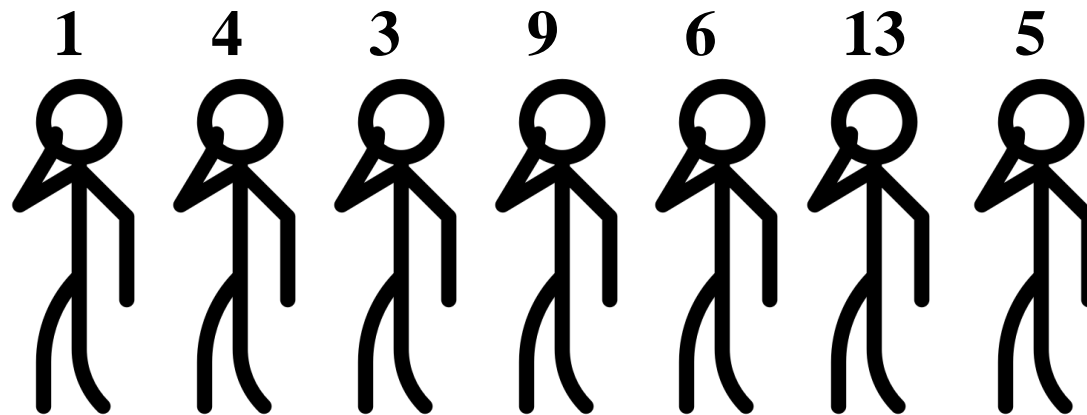
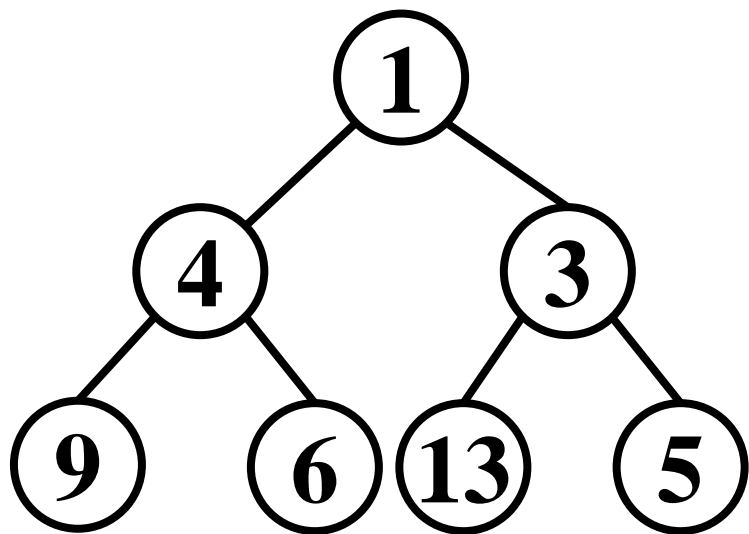
优先队列

- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - $Q.Insert()$



优先队列

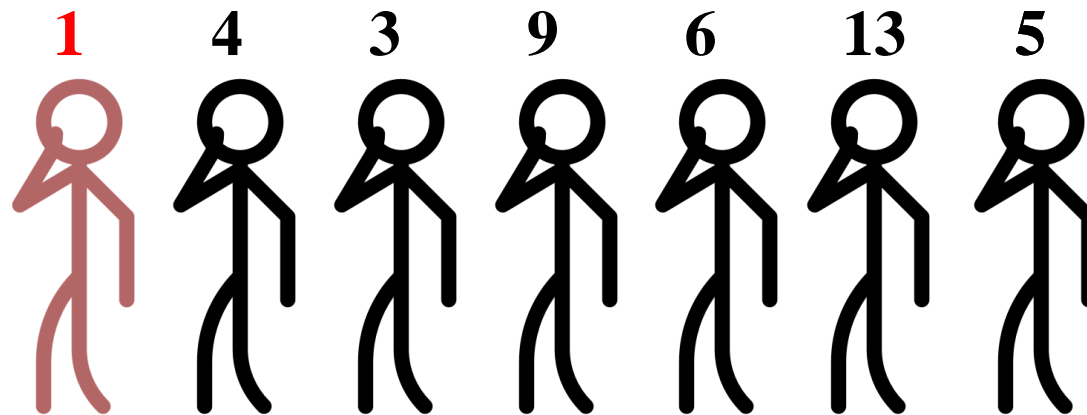
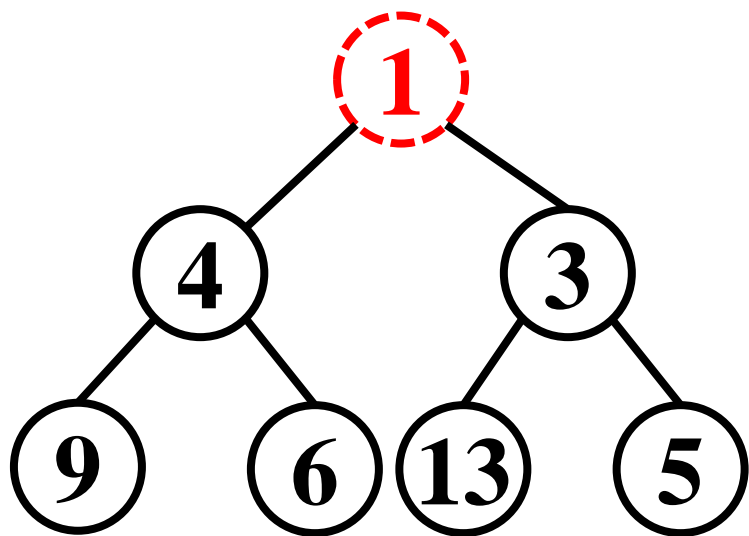
- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - *Q.Insert()*



优先队列 Q

优先队列

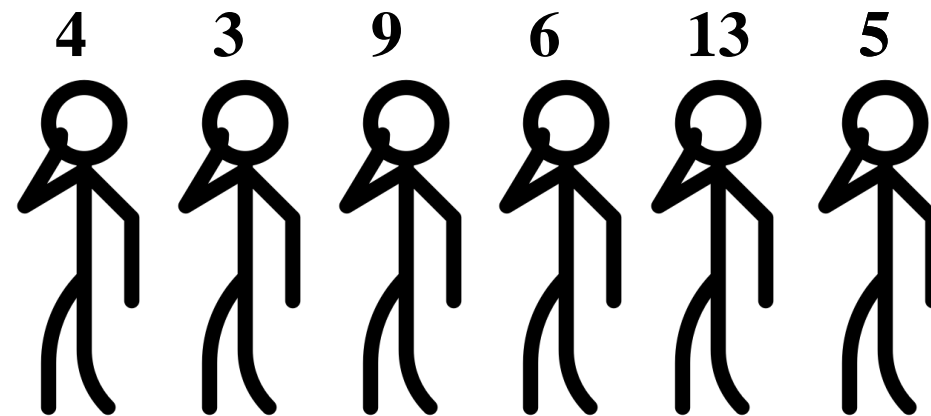
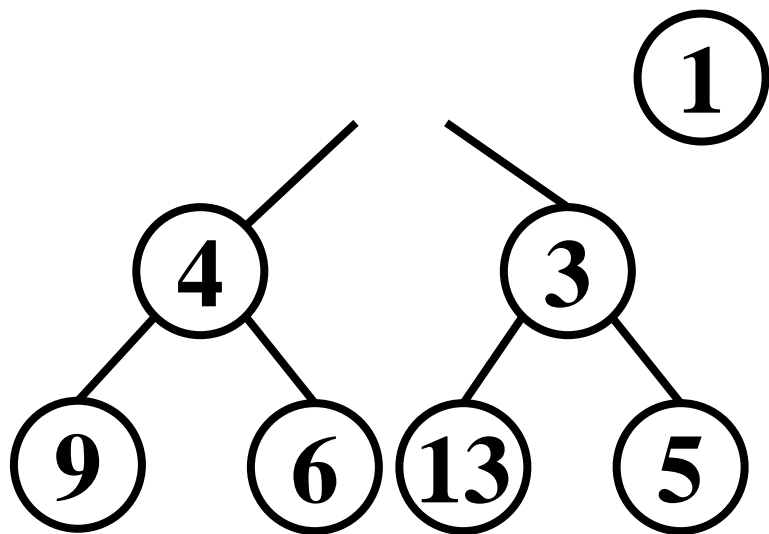
- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - $Q.Insert()$
 - $Q.ExtractMin()$



优先队列 Q

优先队列

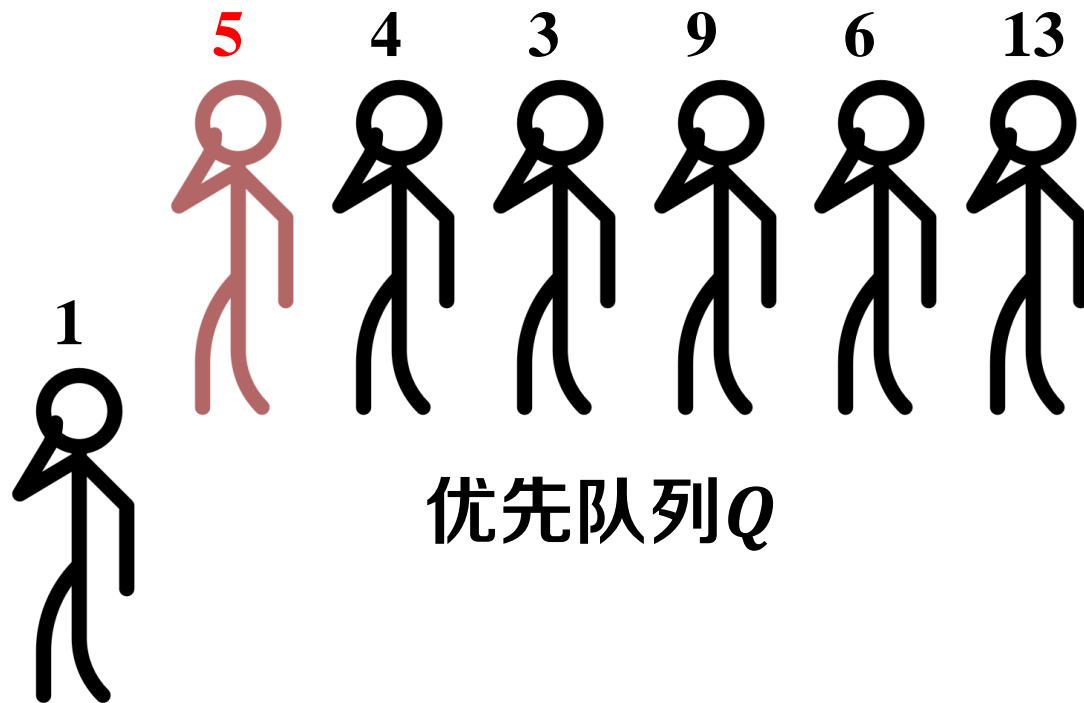
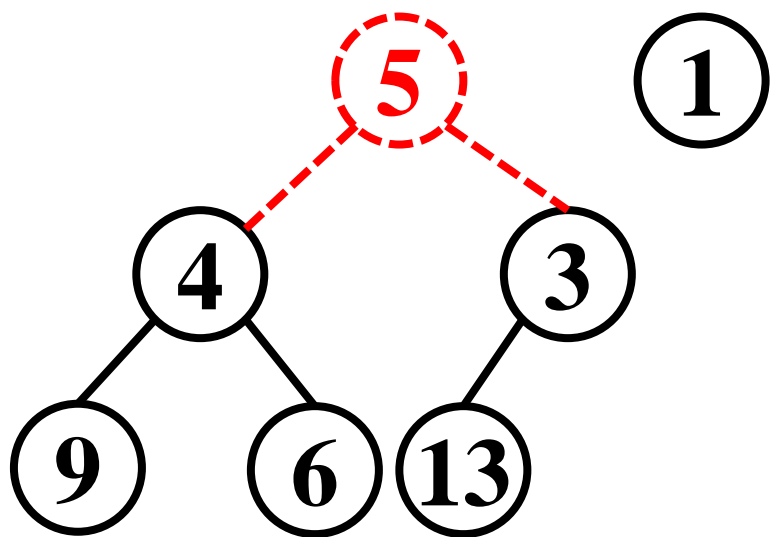
- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - $Q.Insert()$
 - $Q.ExtractMin()$



优先队列 Q

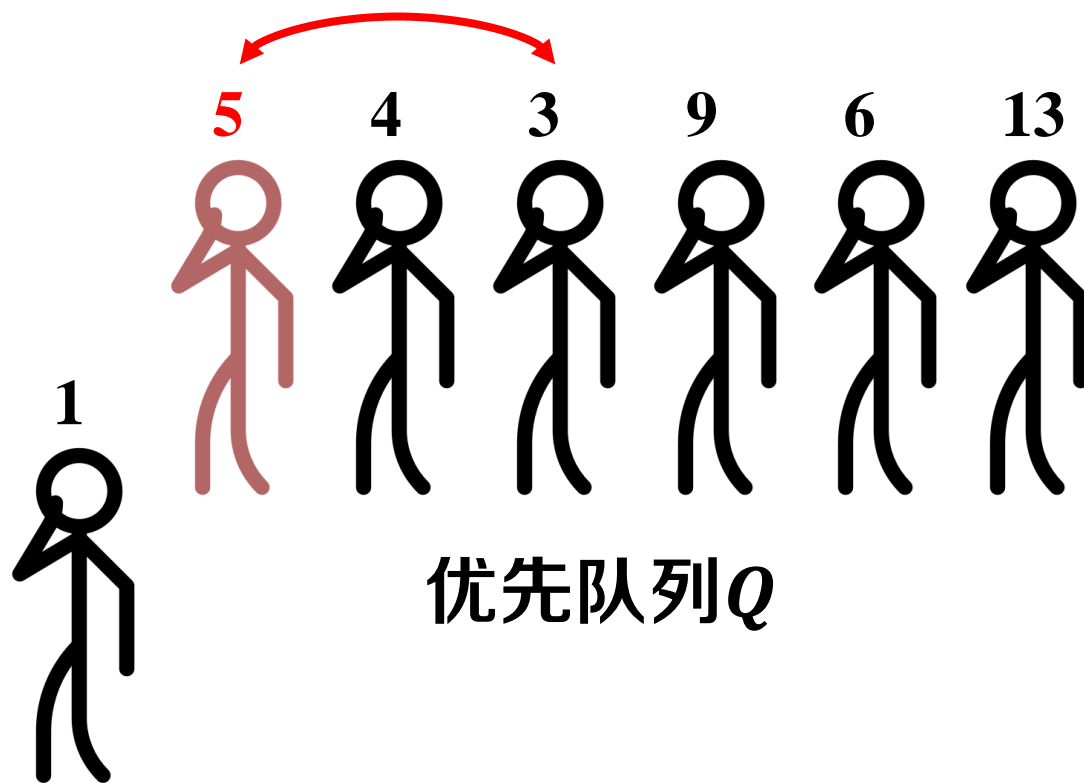
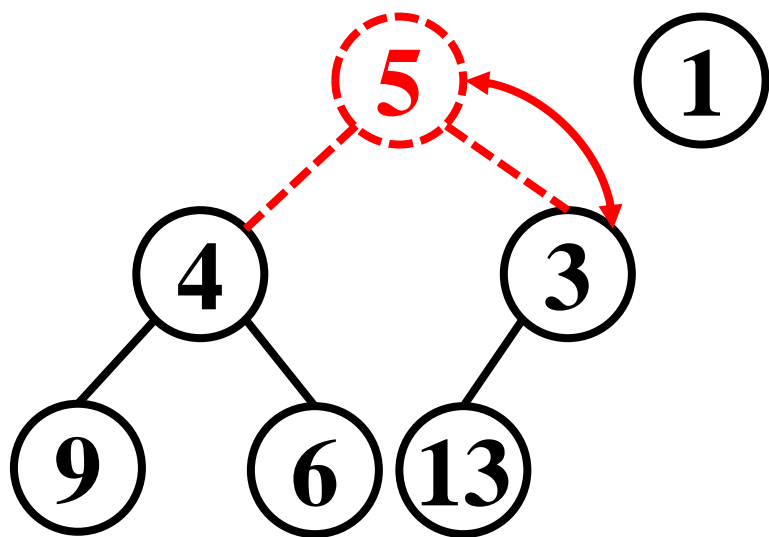
优先队列

- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - $Q.Insert()$
 - $Q.ExtractMin()$



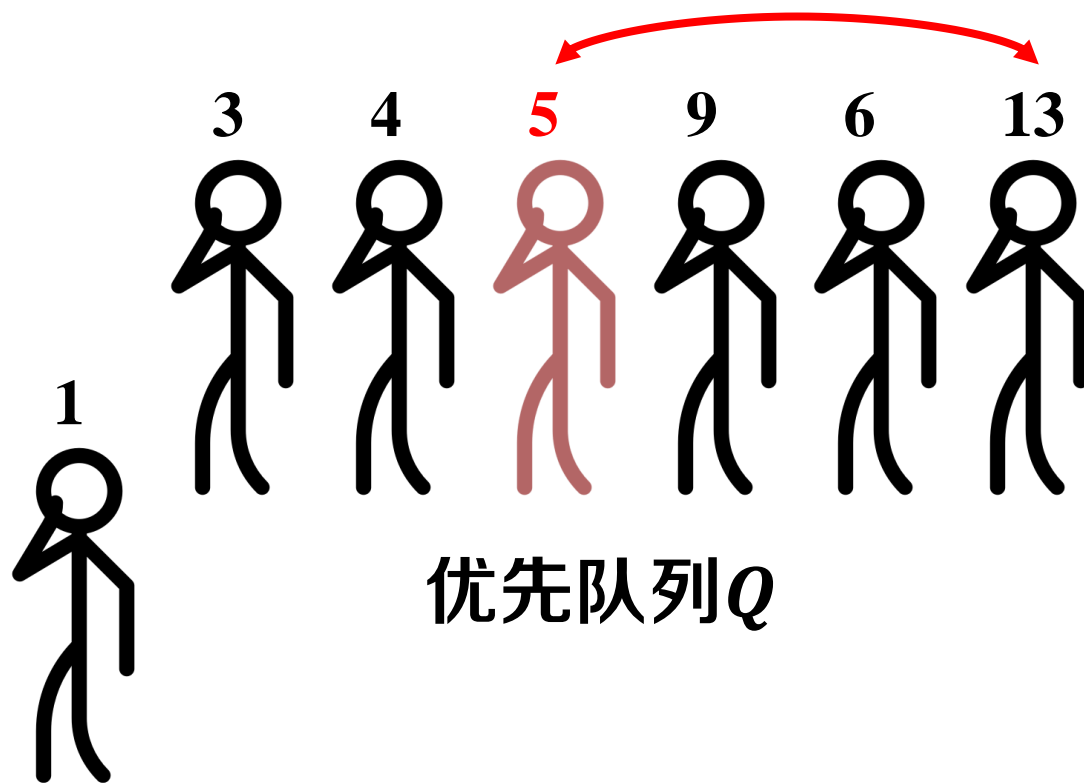
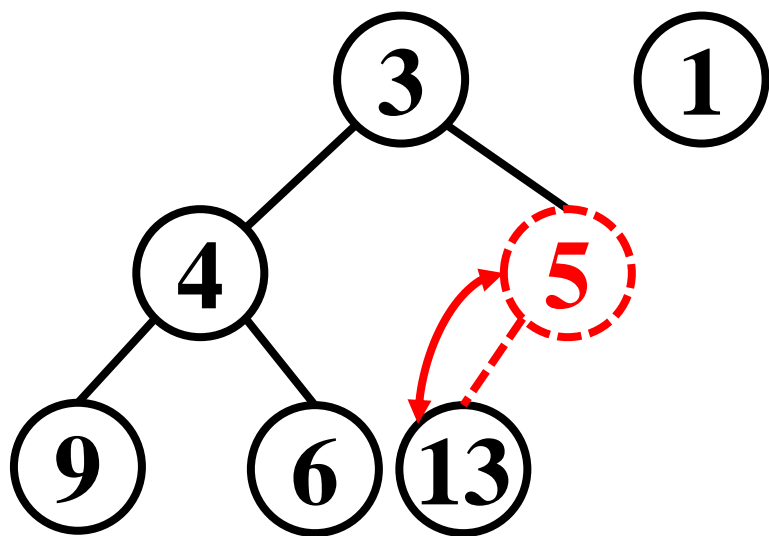
优先队列

- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - $Q.Insert()$
 - $Q.ExtractMin()$



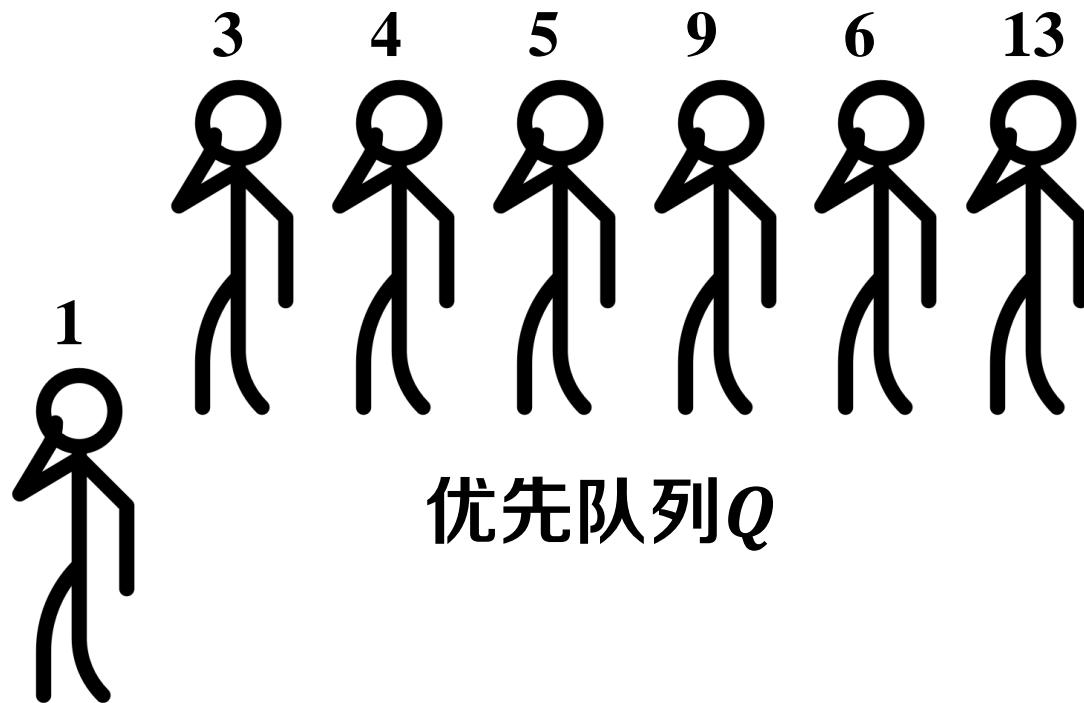
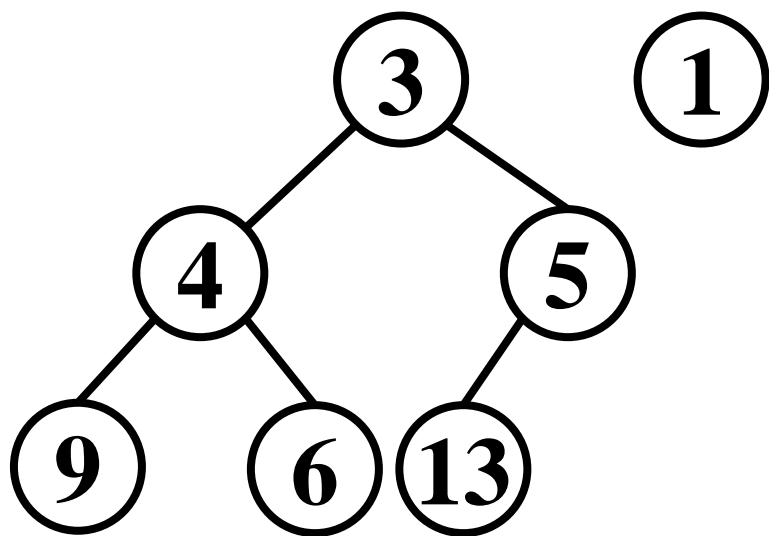
优先队列

- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - *Q.Insert()*
 - *Q.ExtractMin()*



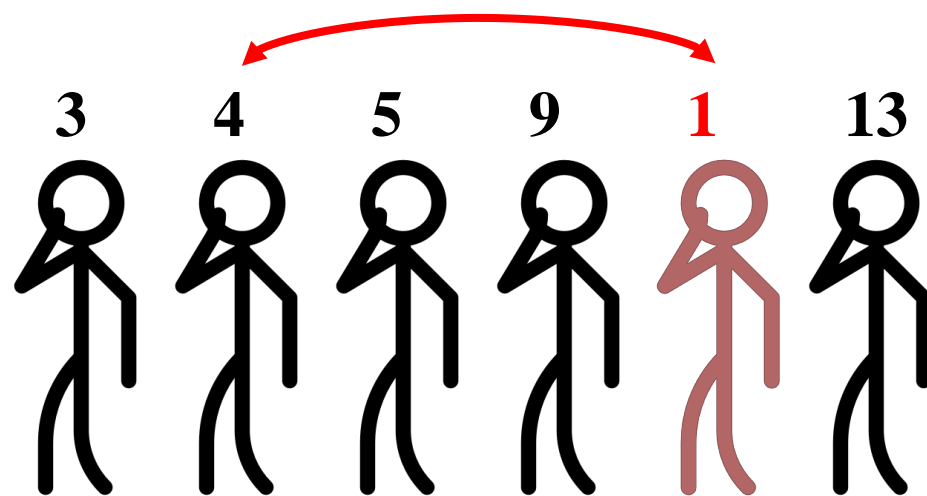
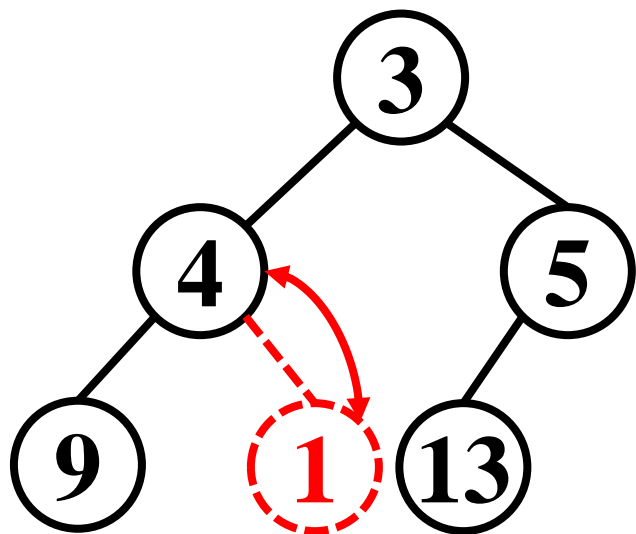
优先队列

- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - *Q.Insert()*
 - *Q.ExtractMin()*



优先队列

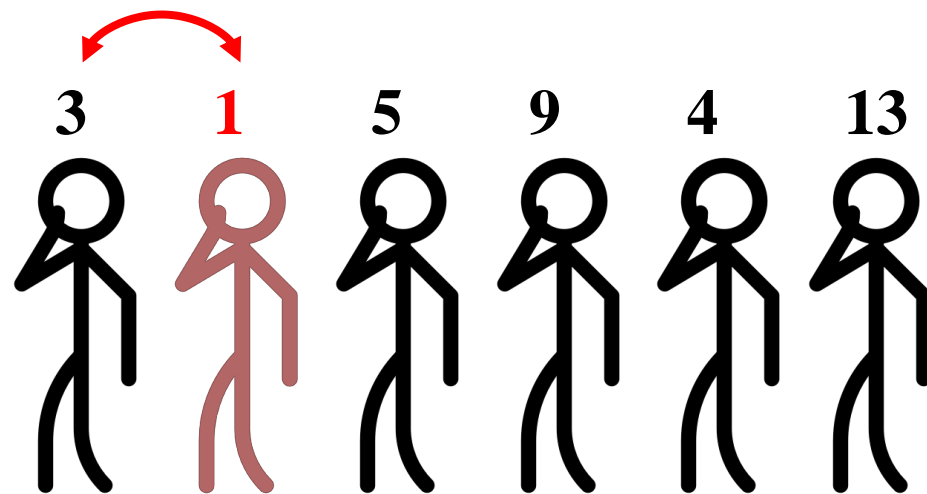
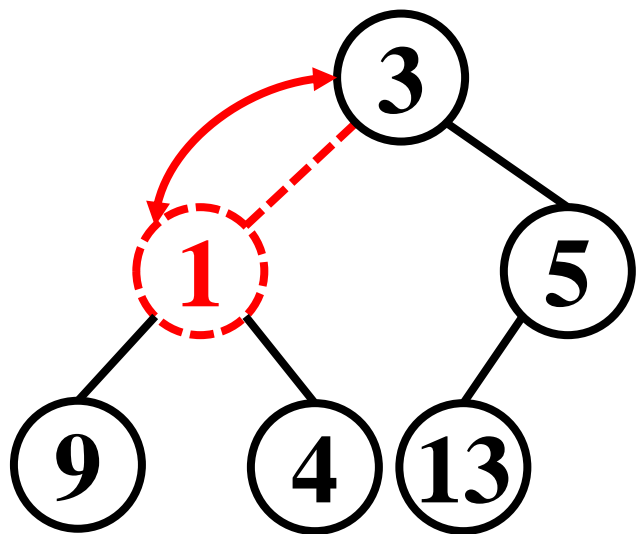
- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - *Q.Insert()*
 - *Q.ExtractMin()*
 - *Q.DecreaseKey()*



优先队列 Q

优先队列

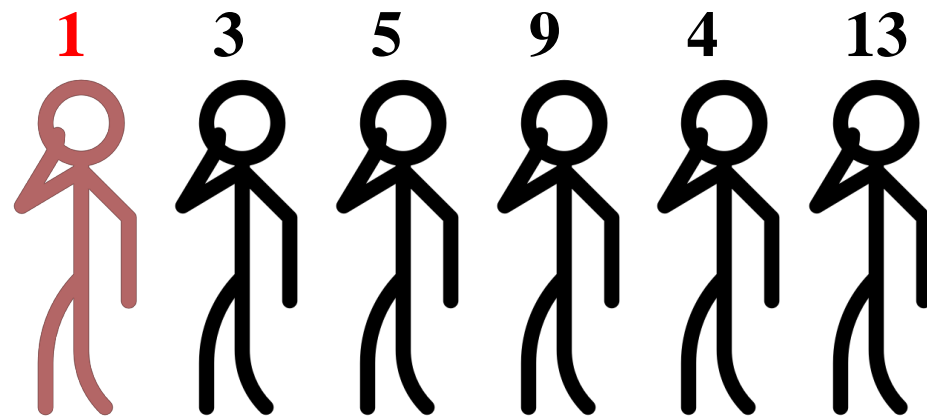
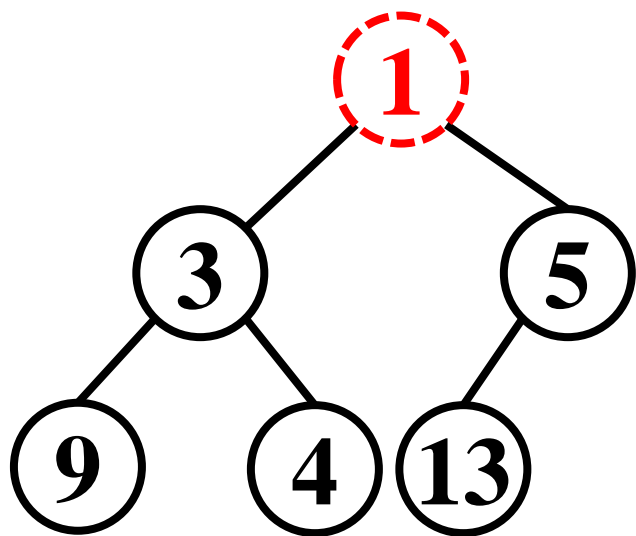
- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - *Q.Insert()*
 - *Q.ExtractMin()*
 - *Q.DecreaseKey()*



优先队列 Q

优先队列

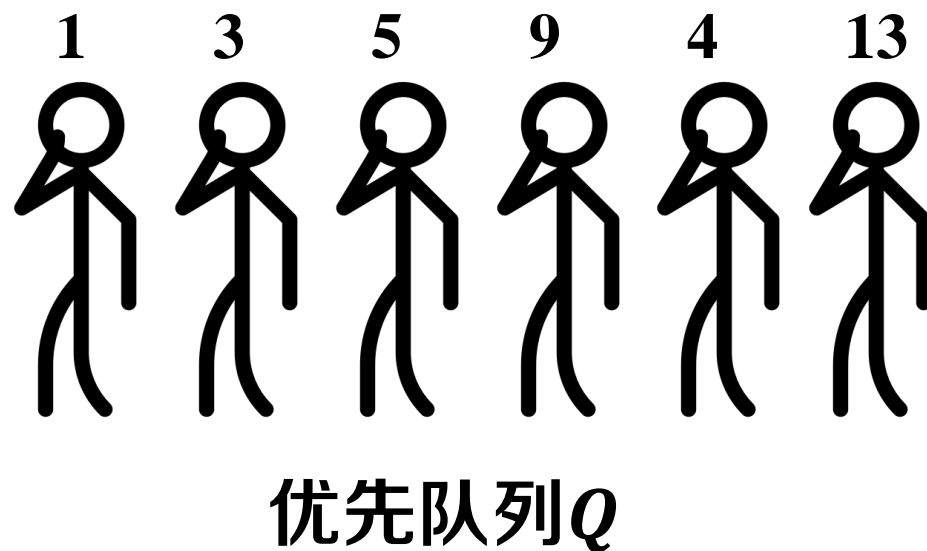
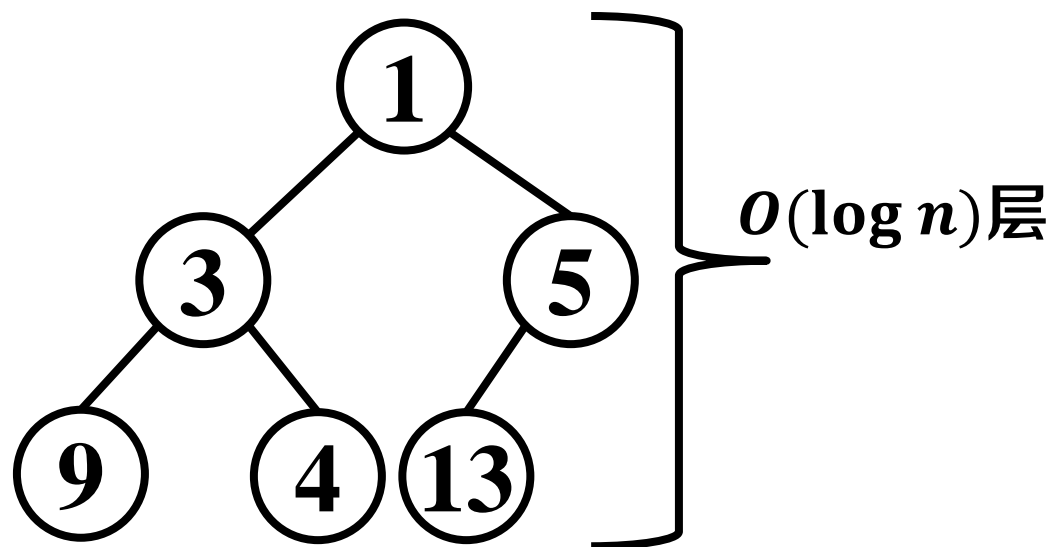
- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - *Q.Insert()*
 - *Q.ExtractMin()*
 - *Q.DecreaseKey()*



优先队列 Q

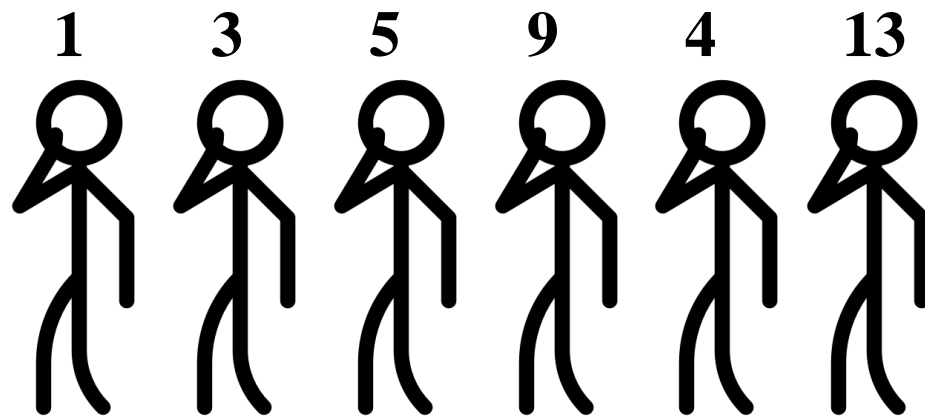
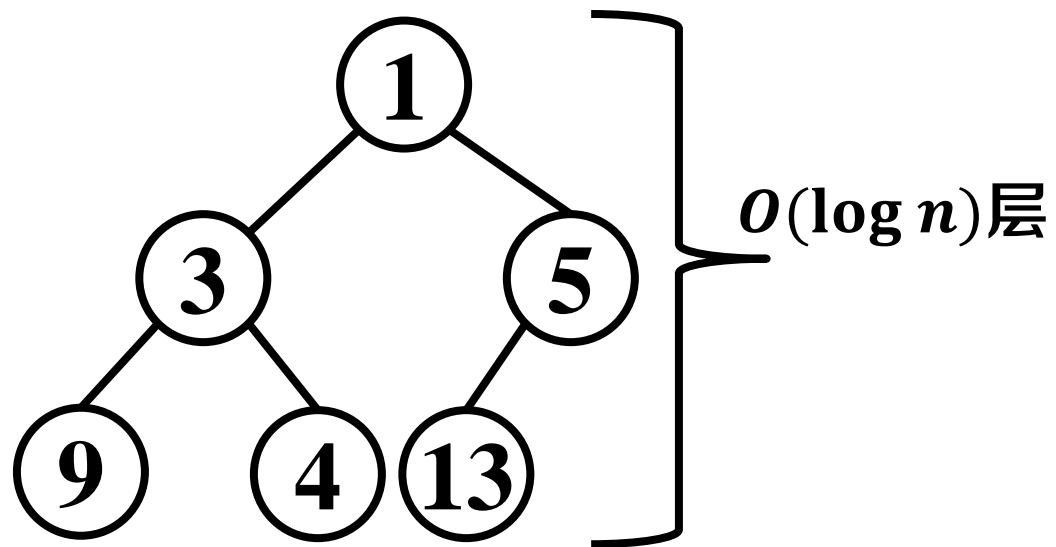
优先队列

- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - *Q.Insert()*
 - *Q.ExtractMin()*
 - *Q.DecreaseKey()*



优先队列

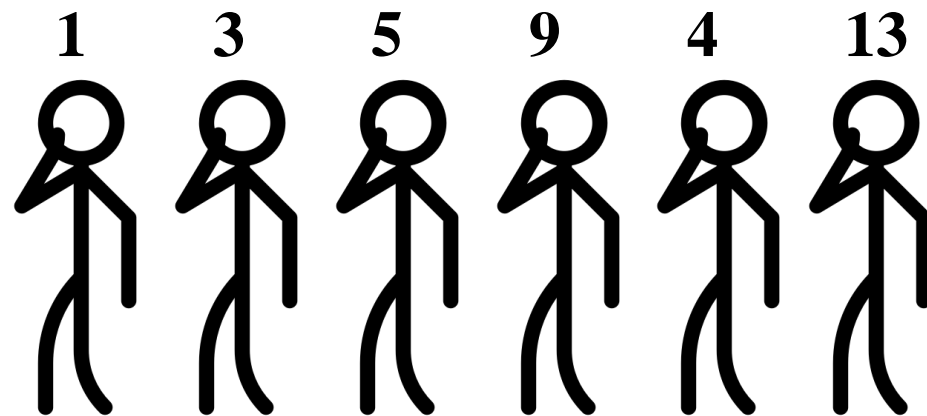
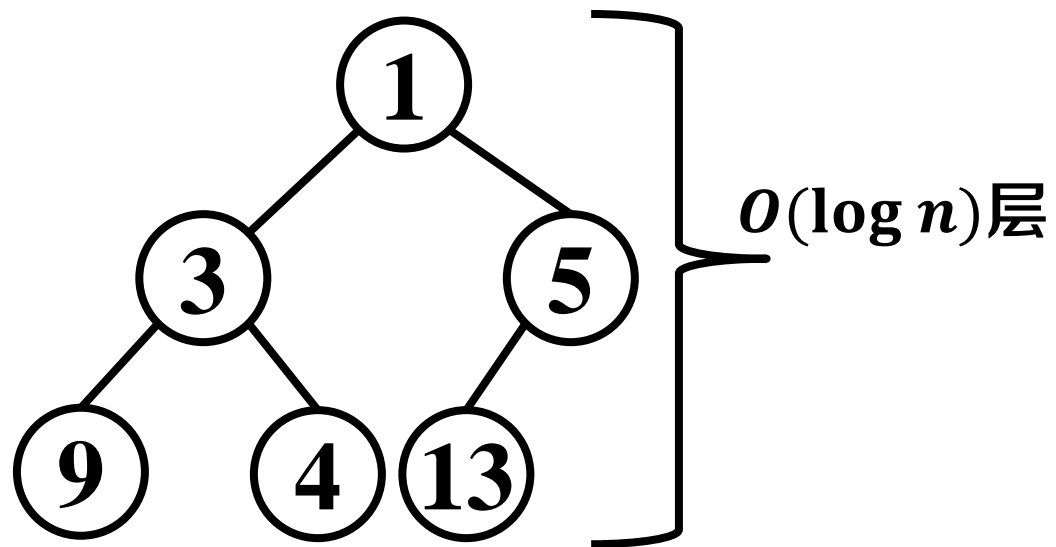
- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - $Q.Insert()$ 时间复杂度 $O(\log n)$
 - $Q.ExtractMin()$ 时间复杂度 $O(\log n)$
 - $Q.DecreaseKey()$ 时间复杂度 $O(\log n)$



优先队列 Q

优先队列

- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
 - $Q.Insert()$ 时间复杂度 $O(\log n)$
 - $Q.ExtractMin()$ 时间复杂度 $O(\log n)$
 - $Q.DecreaseKey()$ 时间复杂度 $O(\log n)$



优先队列 Q

使用优先队列，高效查找的安全边

- MST-Prim-PriQueue(G)

输入: 图 $G = \langle V, E, W \rangle$

输出: 最小生成树 T

新建一维数组 $color[1..|V|]$, $dist[1..|V|]$, $pred[1..|V|]$

新建空优先队列 Q

//初始化

```
for  $u \in V$  do
     $color[u] \leftarrow WHITE$ 
     $dist[u] \leftarrow \infty$ 
     $pred[u] \leftarrow NULL$ 
end
```

初始化辅助数组

$dist[1] \leftarrow 0$

$Q.Insert(V, dist)$

- **MST-Prim-PriQueue(G)**

输入: 图 $G = \langle V, E, W \rangle$

输出: 最小生成树 T

新建一维数组 $color[1..|V|]$, $dist[1..|V|]$, $pred[1..|V|]$

新建空优先队列 Q

//初始化

for $u \in V$ do

$color[u] \leftarrow WHITE$

$dist[u] \leftarrow \infty$

$pred[u] \leftarrow NULL$

end

$dist[1] \leftarrow 0$

$Q.Insert(V, dist)$

选择任意起点

- MST-Prim-PriQueue(G)

输入: 图 $G = \langle V, E, W \rangle$

输出: 最小生成树 T

新建一维数组 $color[1..|V|]$, $dist[1..|V|]$, $pred[1..|V|]$

新建空优先队列 Q

//初始化

for $u \in V$ do

$color[u] \leftarrow WHITE$

$dist[u] \leftarrow \infty$

$pred[u] \leftarrow NULL$

end

$dist[1] \leftarrow 0$

$Q.Insert(V, dist)$

初始化优先队列

- **MST-Prim-PriQueue(G)**

```
//执行最小生成树算法
while 优先队列 $Q$ 非空 do
     $v \leftarrow Q.ExtractMin()$ 
    for  $u \in G.Adj[v]$  do
        if  $color[u] = WHITE$  and  $w(v, u) < dist[u]$  then
             $dist[u] \leftarrow w(v, u)$ 
             $pred[u] \leftarrow v$ 
             $Q.DecreaseKey((u, dist[u]))$ 
        end
    end
     $color[v] \leftarrow BLACK$ 
end
```

依次添加其他顶点

- **MST-Prim-PriQueue(G)**

//执行最小生成树算法

while 优先队列 Q 非空 do

$v \leftarrow Q.ExtractMin()$

for $u \in G.Adj[v]$ do

if $color[u] = WHITE$ **and** $w(v, u) < dist[u]$ **then**

$dist[u] \leftarrow w(v, u)$

$pred[u] \leftarrow v$

$Q.DecreaseKey((u, dist[u]))$

end

end

$color[v] \leftarrow BLACK$

end

选择安全边

- MST-Prim-PriQueue(G)

//执行最小生成树算法

while 优先队列 Q 非空 do

~~$v \leftarrow Q.ExtractMin()$~~

for $u \in G.Adj[v]$ do

if $color[u] = WHITE$ and $w(v, u) < dist[u]$ then

$dist[u] \leftarrow w(v, u)$

$pred[u] \leftarrow v$

$Q.DecreaseKey((u, dist[u]))$

end

~~end~~

$color[v] \leftarrow BLACK$

end

更新距离数组，调整优先队列

- **MST-Prim-PriQueue(G)**

//执行最小生成树算法

while 优先队列 Q 非空 **do**

$v \leftarrow Q.ExtractMin()$

for $u \in G.Adj[v]$ **do**

if $color[u] = WHITE$ **and** $w(v, u) < dist[u]$ **then**

$dist[u] \leftarrow w(v, u)$

$pred[u] \leftarrow v$

$Q.DecreaseKey((u, dist[u]))$

end

end

$color[v] \leftarrow BLACK$

end

标记顶点处理完成

- **MST-Prim-PriQueue(G)**

输入: 图 $G = \langle V, E, W \rangle$

输出: 最小生成树 T

新建一维数组 $color[1..|V|]$, $dist[1..|V|]$, $pred[1..|V|]$

新建空优先队列 Q

//初始化

for $u \in V$ do

$color[u] \leftarrow WHITE$

$dist[u] \leftarrow \infty$

$pred[u] \leftarrow NULL$

end

$dist[1] \leftarrow 0$

$Q.Insert(V, dist)$

$O(|V|)$

- MST-Prim-PriQueue(G)

//执行最小生成树算法

while 优先队列 Q 非空 do

$v \leftarrow Q.ExtractMin()$

----- $O(\log|V|)$

 for $u \in G.Adj[v]$ do

 if $color[u] = WHITE$ and $w(v, u) < dist[u]$ then

$dist[u] \leftarrow w(v, u)$

$pred[u] \leftarrow v$

$Q.DecreaseKey((u, dist[u]))$ ----- $O(\log|V|)$

 end

 end

$color[v] \leftarrow BLACK$

end

- MST-Prim-PriQueue(G)

//执行最小生成树算法

while 优先队列 Q 非空 do

$v \leftarrow Q.ExtractMin()$

 for $u \in G.Adj[v]$ do

 if $color[u] = WHITE$ and $w(v, u) < dist[u]$ then

$dist[u] \leftarrow w(v, u)$

$pred[u] \leftarrow v$

$Q.DecreaseKey((u, dist[u]))$

 end

 end

$color[v] \leftarrow BLACK$

end

----- $O(\log|V|)$

$O(\deg(u)\log|V|)$

----- $O(\log|V|)$

- MST-Prim-PriQueue(G)

//执行最小生成树算法

while 优先队列 Q 非空 do

$v \leftarrow Q.ExtractMin()$

 for $u \in G.Adj[v]$ do

 if $color[u] = WHITE$ and $w(v, u) < dist[u]$ then

$dist[u] \leftarrow w(v, u)$

$pred[u] \leftarrow v$

$Q.DecreaseKey((u, dist[u]))$ $--- O(\log|V|)$

 end

 end

$color[v] \leftarrow BLACK$

end

$O(\log|V|)$

$O(|V|\log|V|)$

$O(\deg(u)\log|V|)$

$--- O(\log|V|)$

- MST-Prim-PriQueue(G)

//执行最小生成树算法

while 优先队列 Q 非空 do

$v \leftarrow Q.ExtractMin()$

 for $u \in G.Adj[v]$ do

 if $color[u] = WHITE$ and $w(v, u) < dist[u]$ then

$dist[u] \leftarrow w(v, u)$

$pred[u] \leftarrow v$

$Q.DecreaseKey((u, dist[u]))$ $--- O(\log|V|)$

 end

 end

$color[v] \leftarrow BLACK$

end

$O(\log|V|)$

$O(\deg(u)\log|V|)$

$O(|V|\log|V|)$

$O(|E|\log|V|)$

$$\sum_{u \in V} \deg(u) = 2|E|$$

- MST-Prim-PriQueue(G)

//执行最小生成树算法

while 优先队列 Q 非空 do

$v \leftarrow Q.ExtractMin()$

 for $u \in G.Adj[v]$ do

 if $color[u] = WHITE$ and $w(v, u) < dist[u]$ then

$dist[u] \leftarrow w(v, u)$

$pred[u] \leftarrow v$

$Q.DecreaseKey((u, dist[u]))$

 end

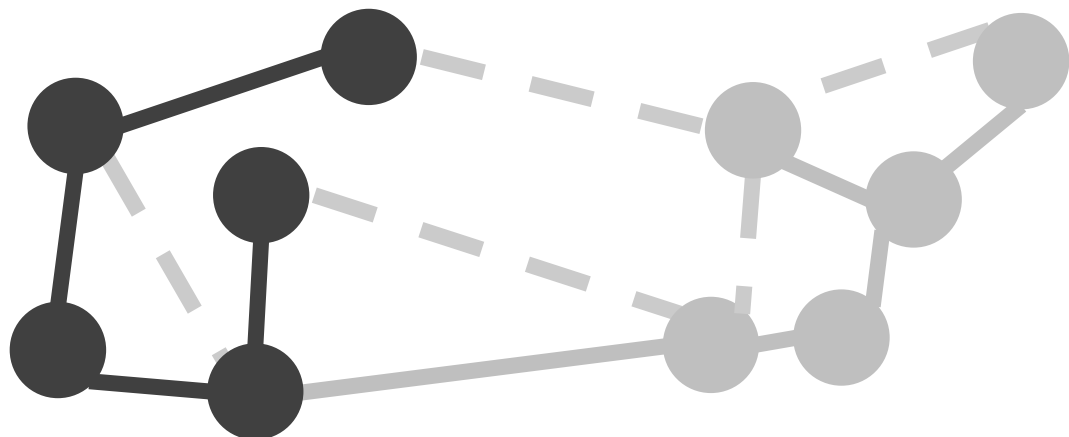
 end

$color[v] \leftarrow BLACK$

end

$O(|E| \cdot \log|V|)$

通用框架	Prim算法
判断是否成环	保持树的结构



通用框架	Prim算法
判断是否成环	保持树的结构
高效寻找轻边	使用优先队列

