Design and Analysis of Algorithms

Part IV: Graph Algorithms

Lecture 27: Minimum Spanning Trees: Prim

童咏昕

北京航空航天大学
计算机学院

图算法篇概述

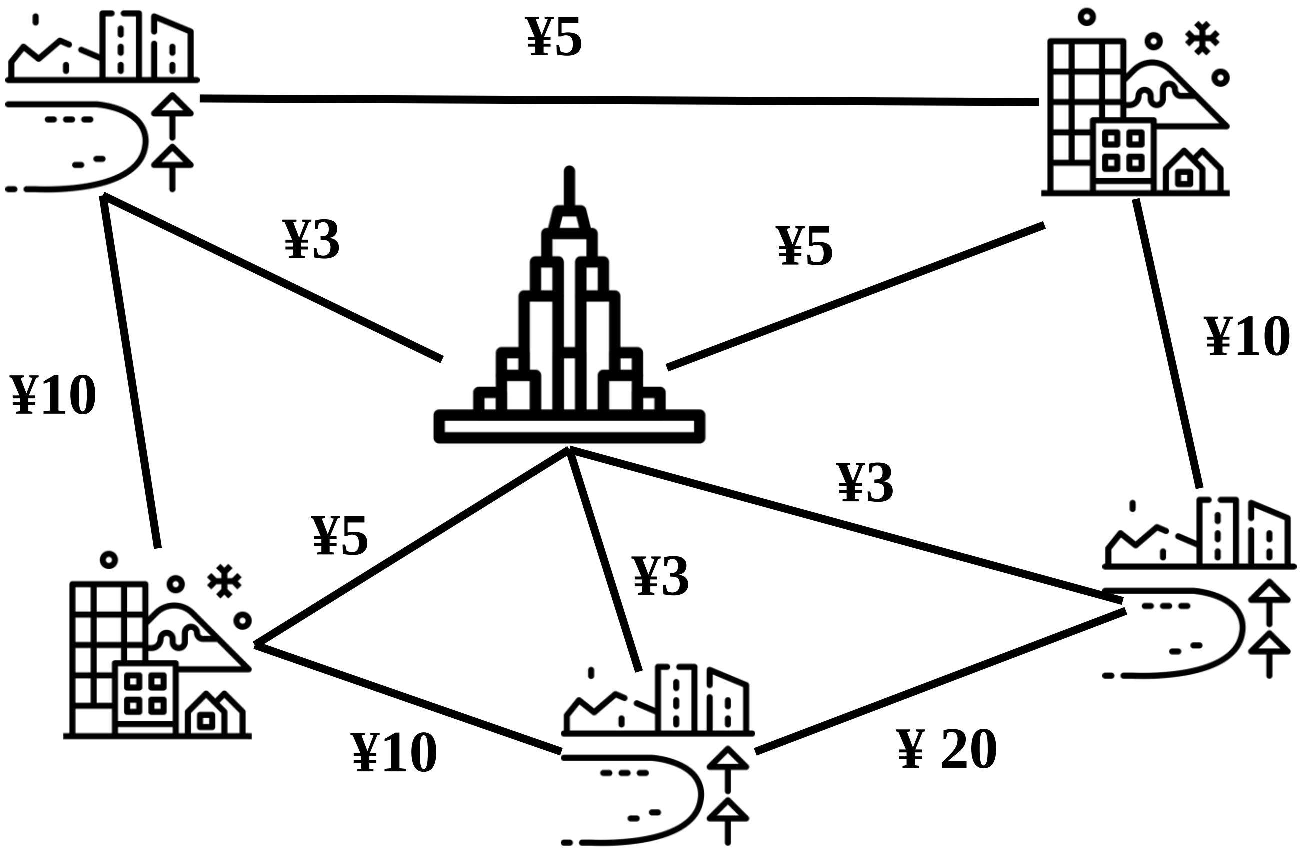
- 在算法课程第四部分“图算法”主题中，我们将主要聚焦于如下经典问题：
  - Basic Concepts in Graph Algorithms（图算法的基本概念）
  - Breadth-First Search (BFS，广度优先搜索）
  - Depth-First Search (DFS，深度优先搜索）
  - Cycle Detection（环路检测）
  - Topological Sort （拓扑排序）
  - Strongly Connected Components（强连通分量）
  - Minimum Spanning Trees (最小生成树)
  - Single Source Shortest Path (单源最短路径)
  - All-Pairs Shortest Paths (所有点对最短路径)
  - Bipartite Graph Matching（二分图匹配）
  - Maximum/Network Flows (最大流/网络流)

提纲

- 问题背景
- 通用框架
- Prim算法
- 算法实例
- 算法分析

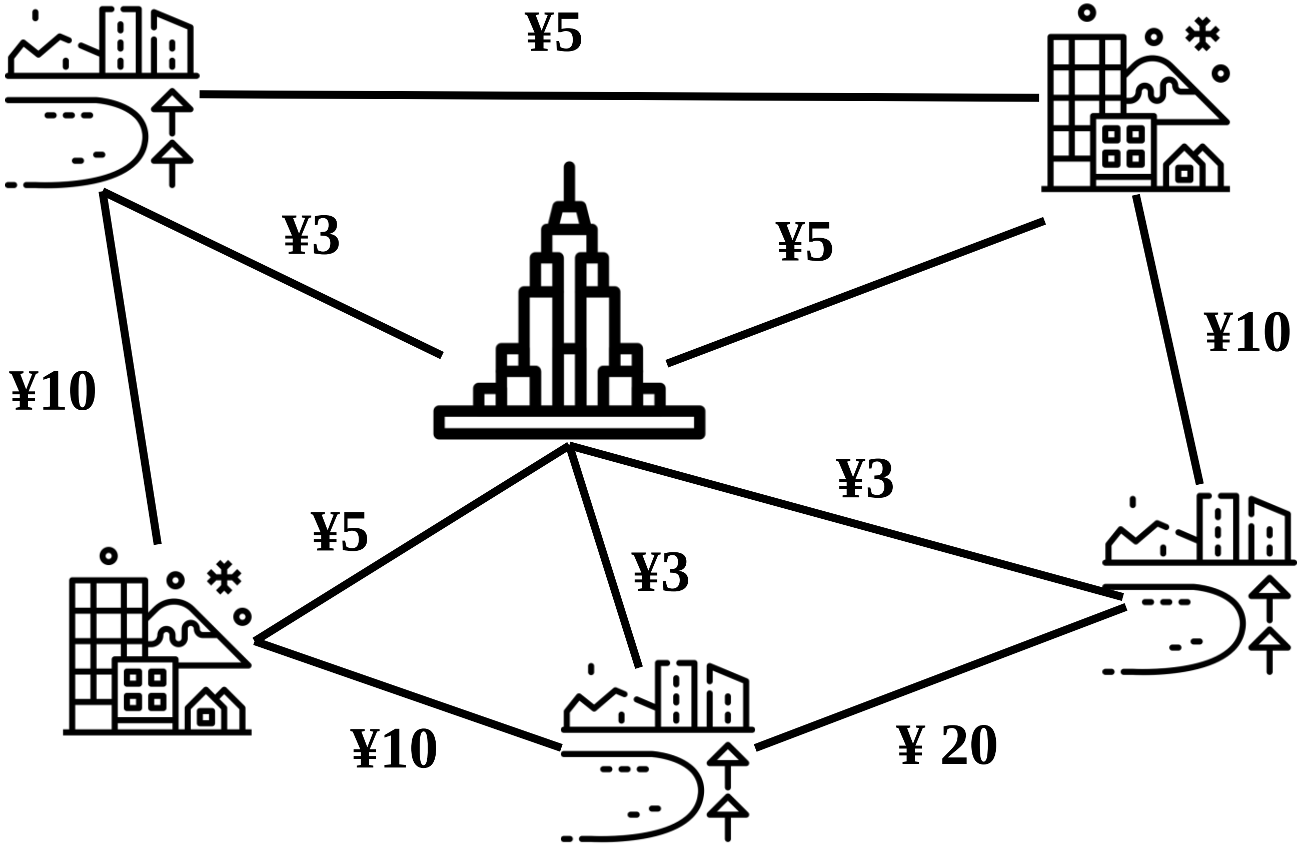
问题背景：道路修建

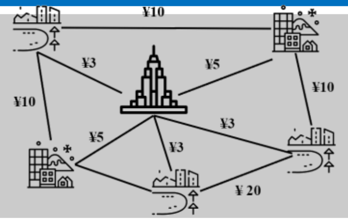
- 需要修建道路连通城市，各道路花费不同



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

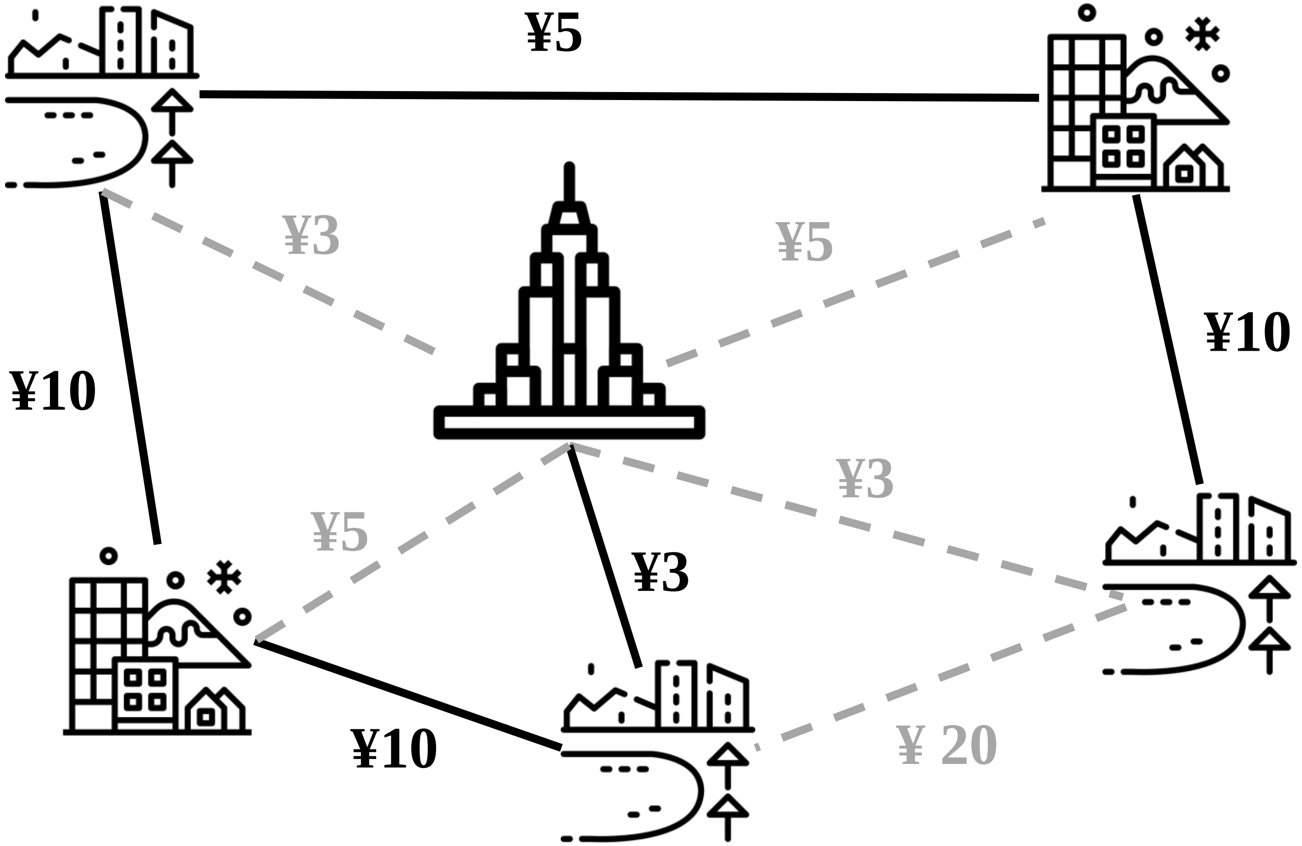


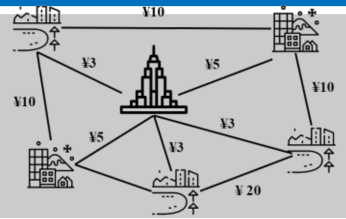
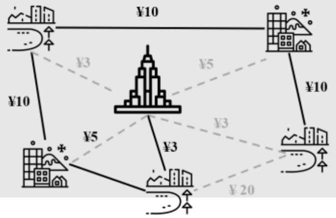
方案	花费
	¥74

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

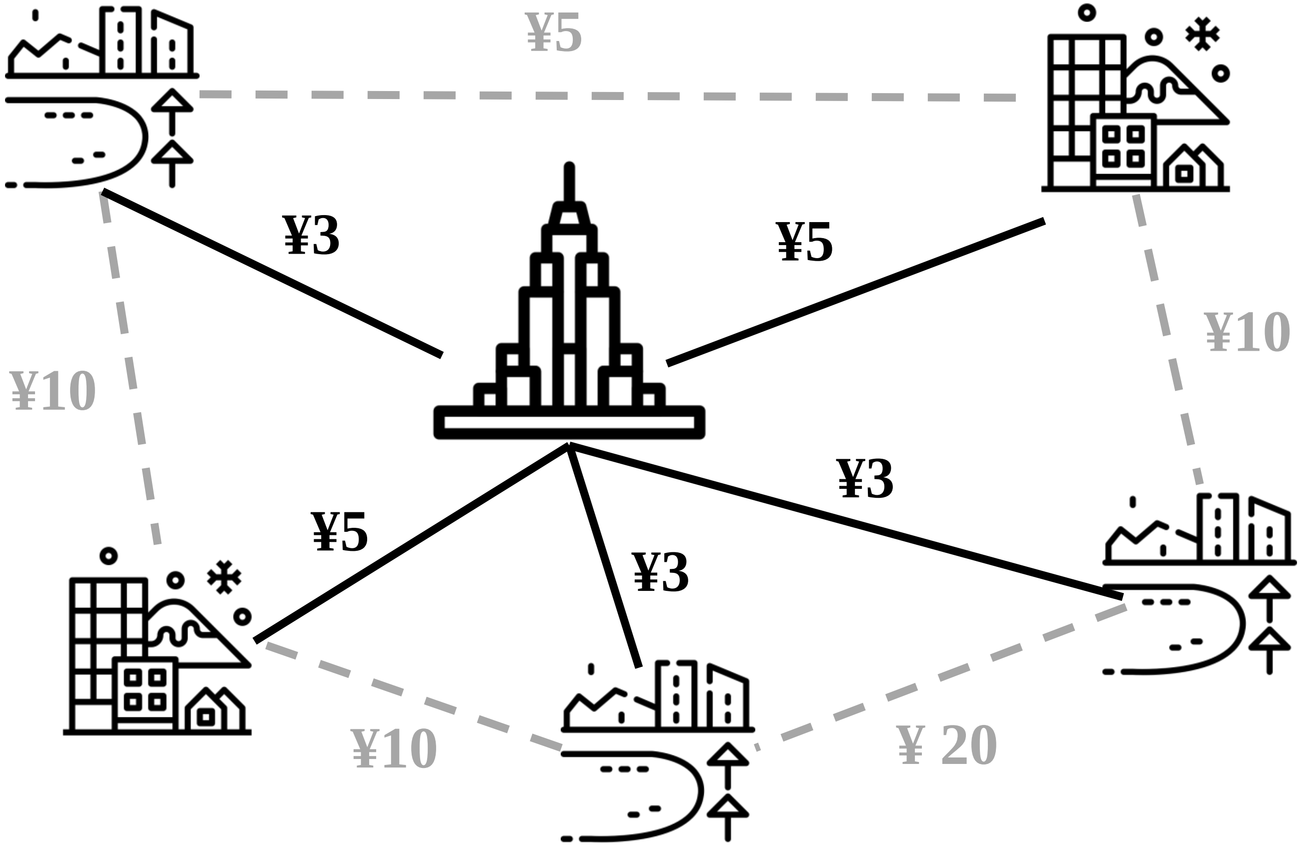


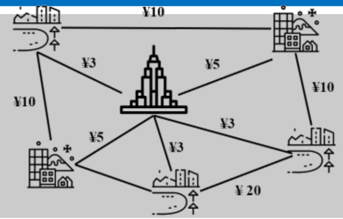
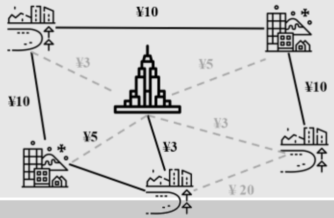
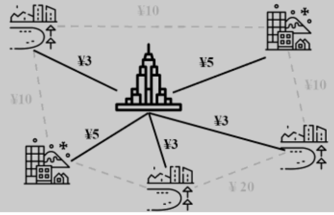
方案	花费
	¥74
	¥38

花费：$10 + 10 + 10 + 5 + 3 = 38$

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

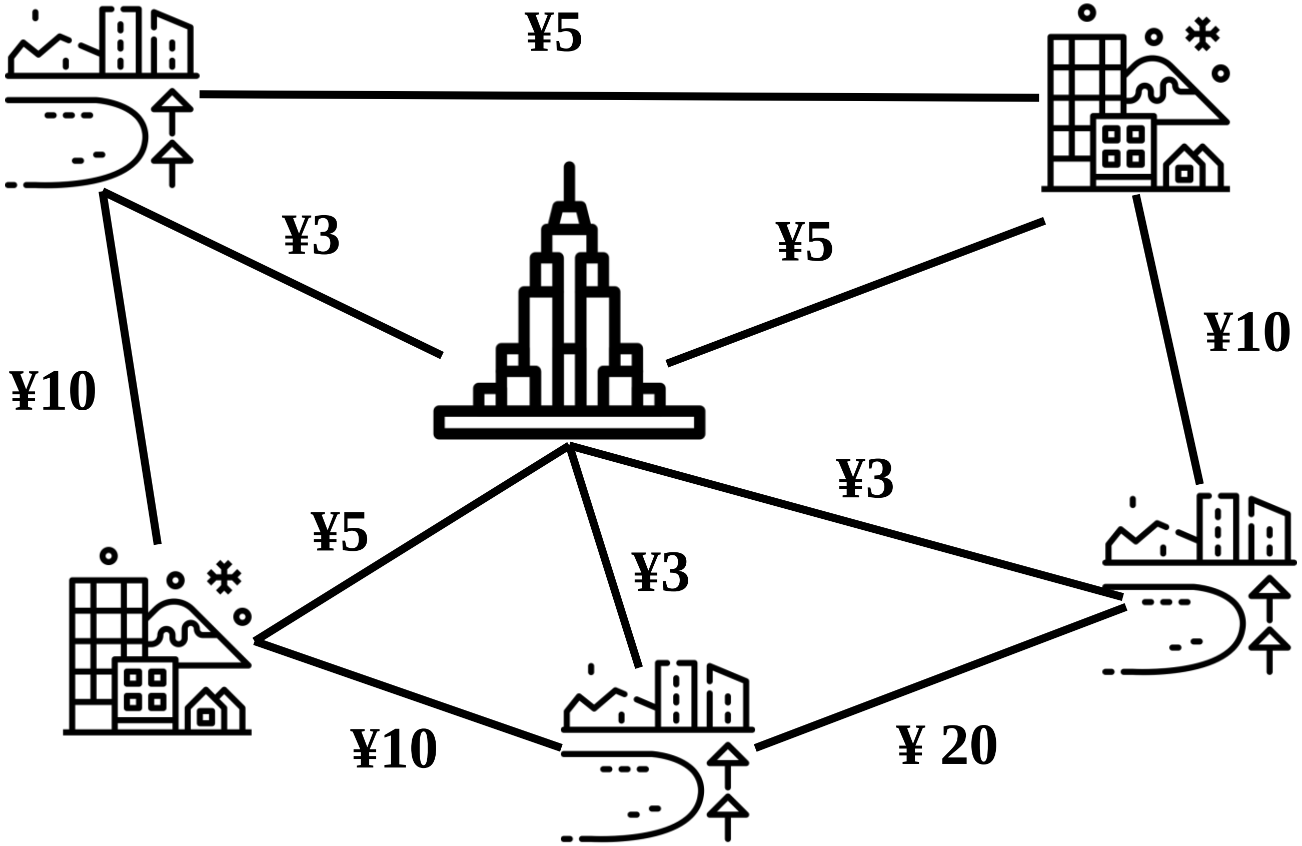


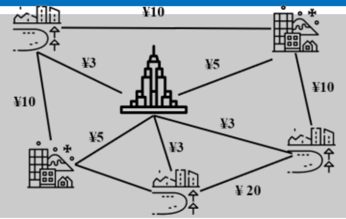
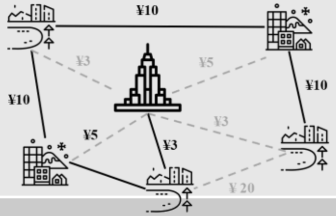
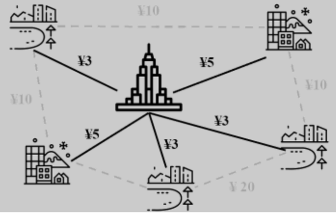
方案	花费
	¥74
	¥38
	¥19

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

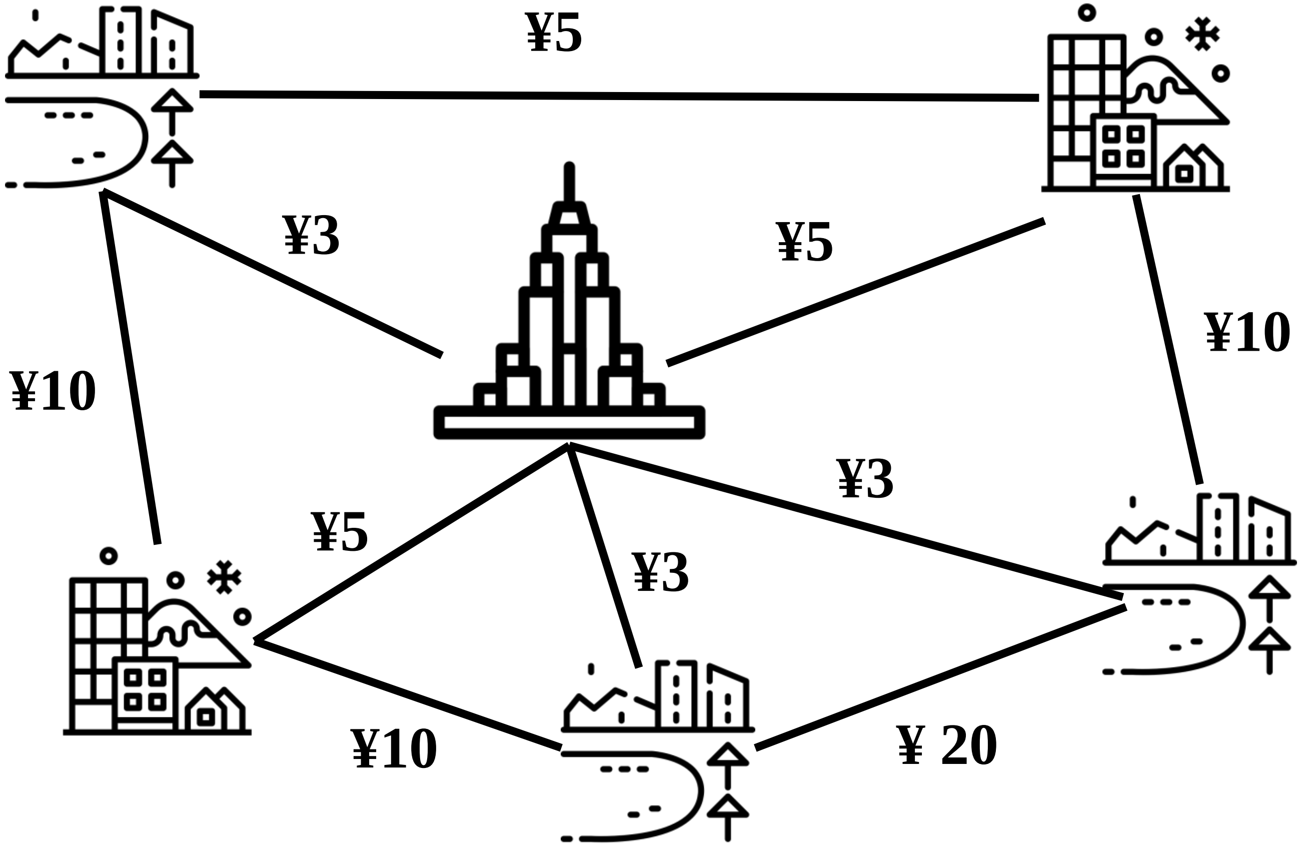


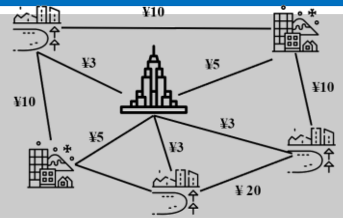
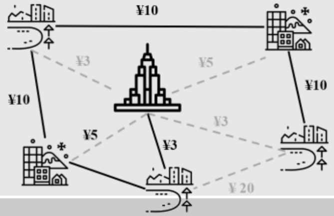
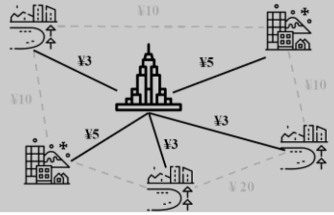
方案	花费
	¥74
	¥38
	¥19

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



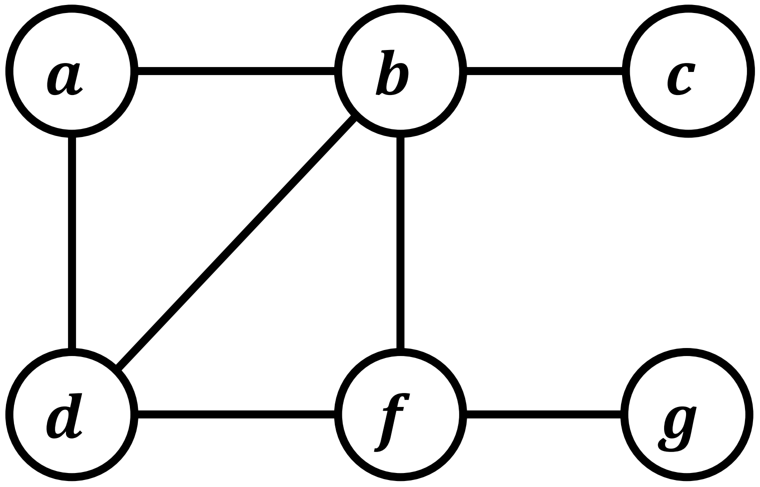
方案	花费
	¥74
	¥38
	¥19

问题：连通各城市的最小花费是多少？

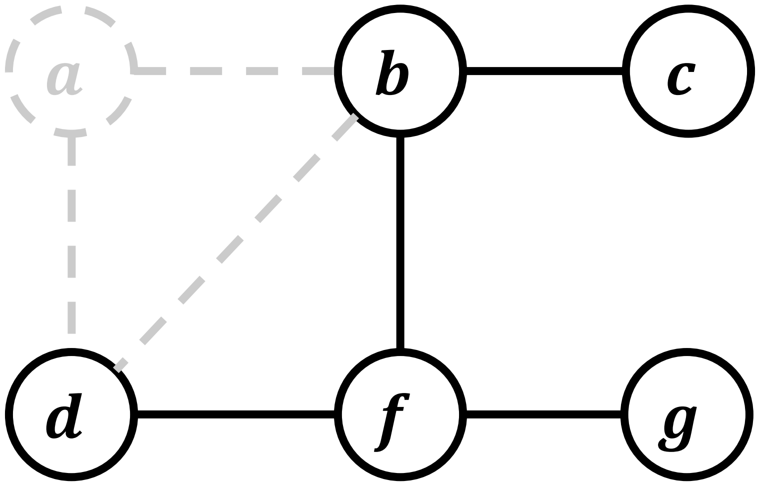
权重最小的连通生成子图

图的概念回顾：生成子图

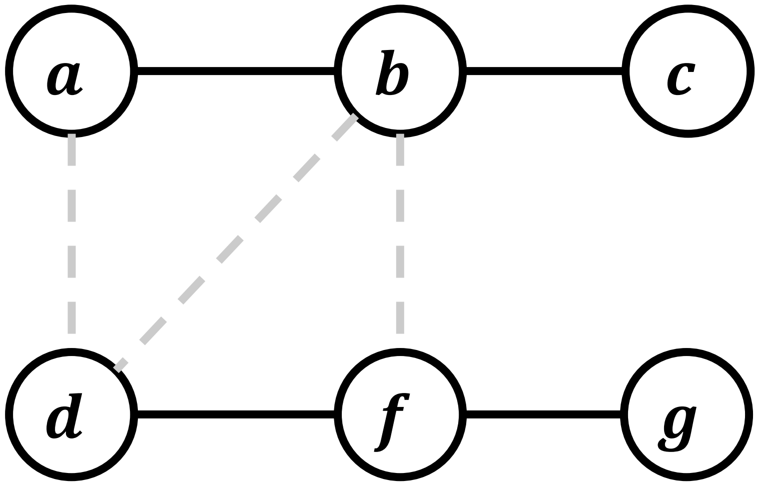
- 子图(Subgraph)
  - 如果$V' \subseteq V, E' \subseteq \mathrm{E}$，则称图$G' =< V', E' >$是图$G$的一个子图
- 生成子图(Spanning Subgraph)
  - 如果$V' = V, E' \subseteq \mathrm{E}$，则称图$G' =< V', E' >$是图$G$的一个生成子图



图G



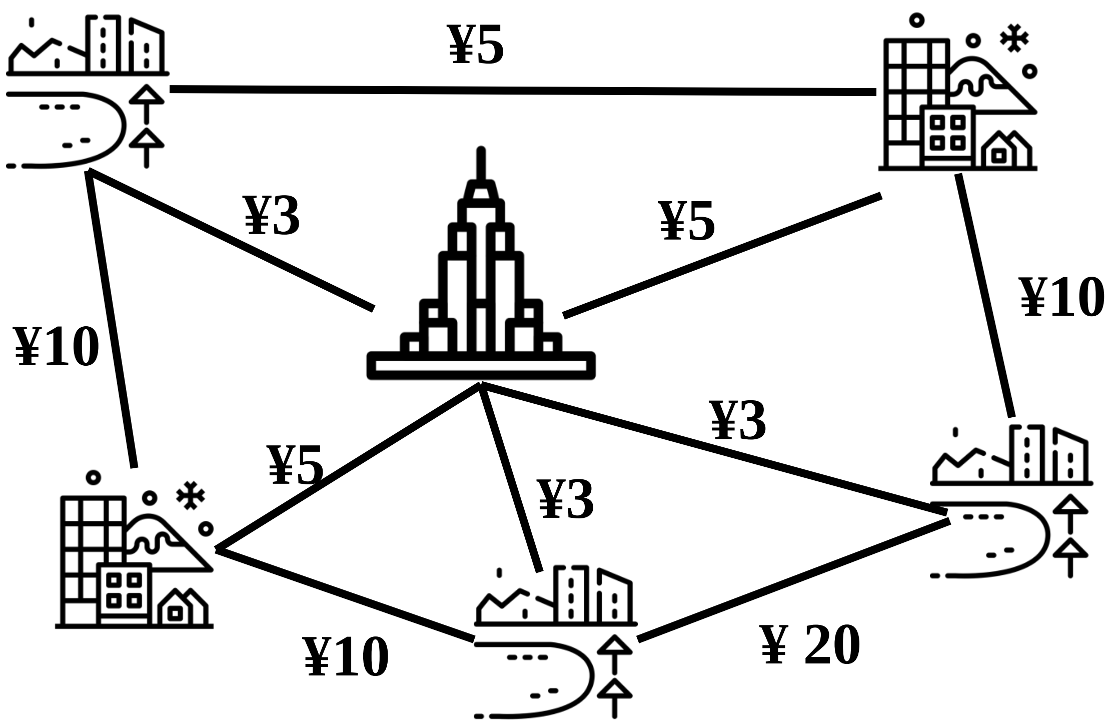
G的子图

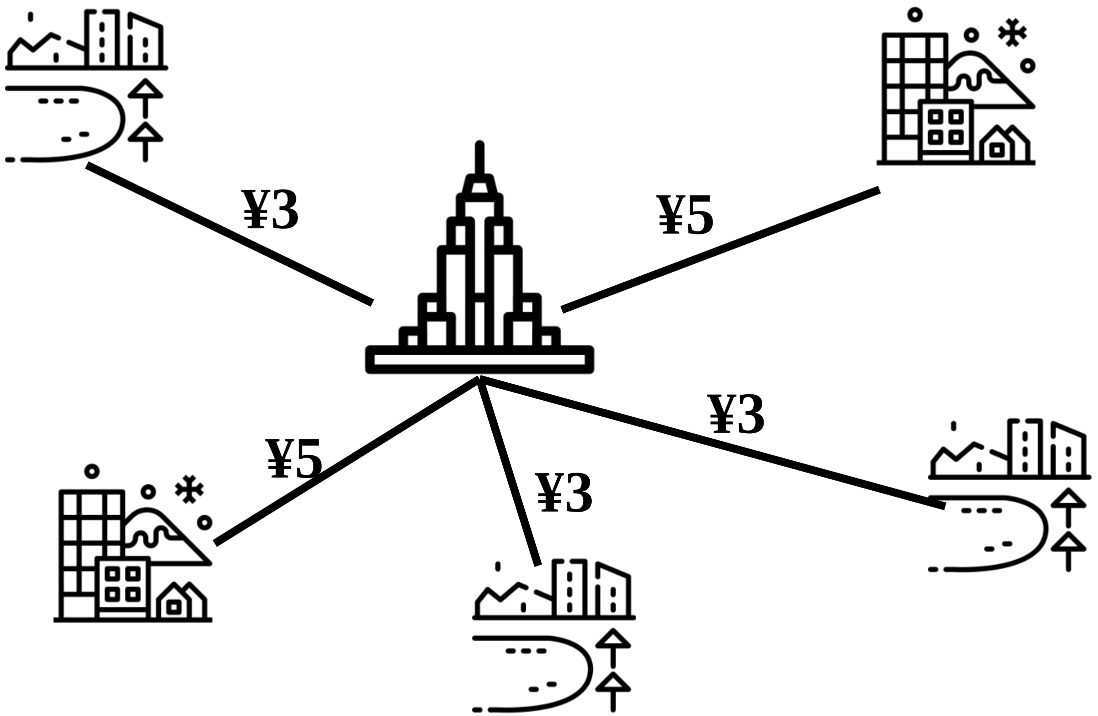


G的生成子图

图的概念：生成树

- 生成树(Spanning Tree)
  - 图$T' = < V', E' >$是无向图G的一个生成子图，并且是连通、无环路的(树)



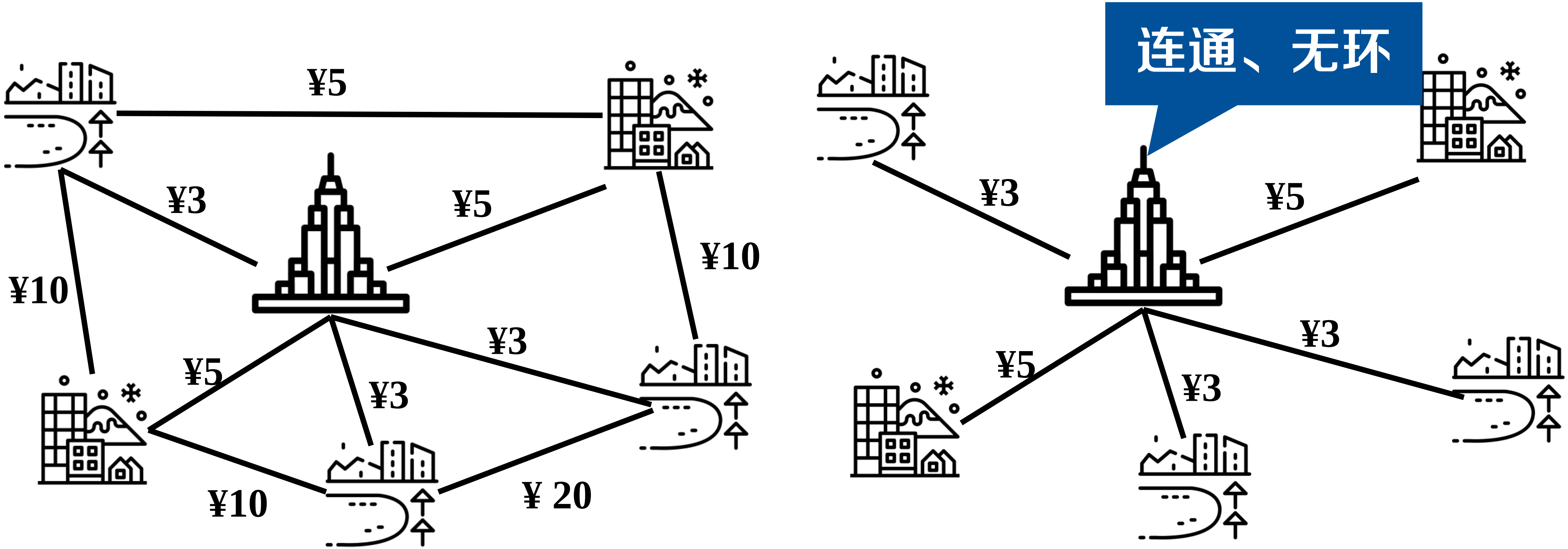


问题：连通各城市的最小花费是多少？

权重最小的连通生成子图

图的概念：生成树

- 生成树(Spanning Tree)
  - 图$T' =< V', E' >$是无向图G的一个生成子图，并且是连通、无环路的(树)

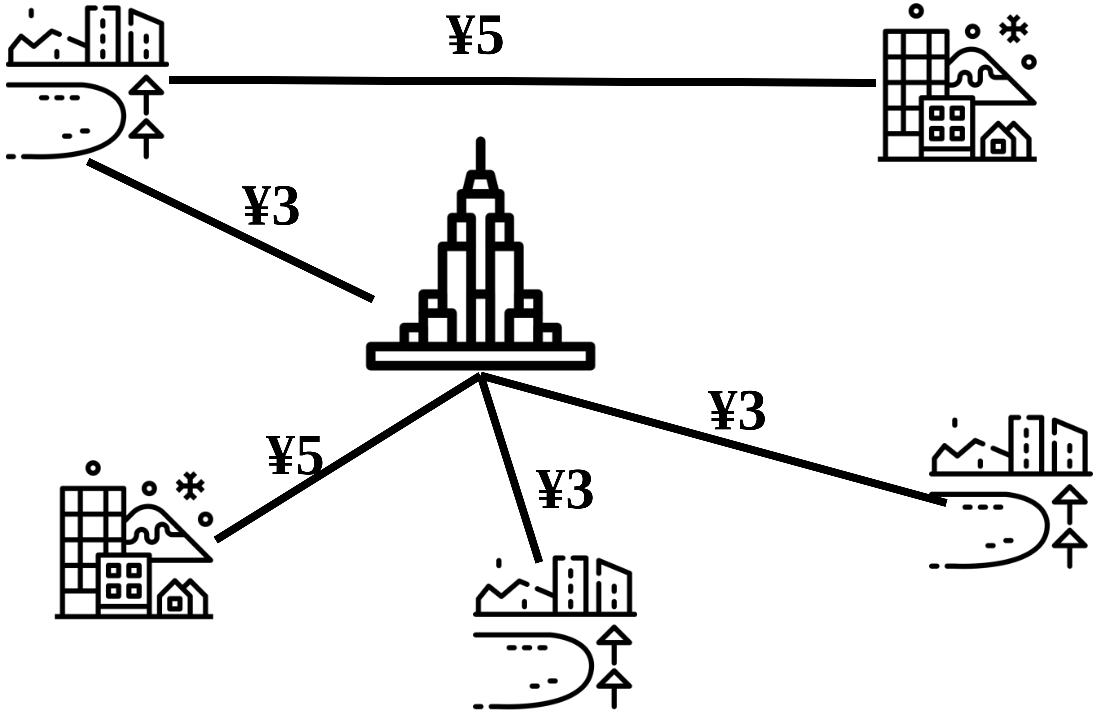


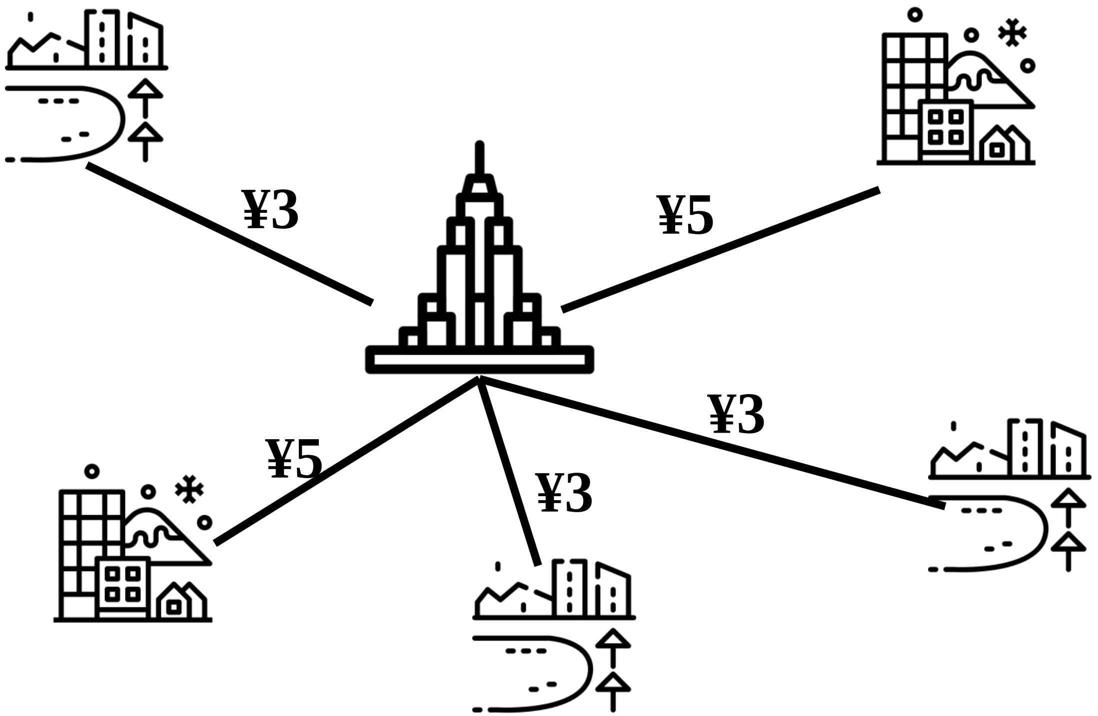
问题：连通各城市的最小花费是多少？

权重最小的生成树

图的概念：生成树

- **生成树(Spanning Tree)**
  - 图$T' = < V', E' >$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！

问题定义

最小生成树问题

Minimum Spanning Tree Problem

输入

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

问题定义

最小生成树问题

Minimum Spanning Tree Problem

输入

- 连通无向图$G=<V,E,W>$, 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$T=<V_T,E_T>$

问题定义

最小生成树问题

Minimum Spanning Tree Problem

输入

- 连通无向图$G =< V, E, W >$, 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$T =< V_T, E_T >$

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

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

问题定义

最小生成树问题

Minimum Spanning Tree Problem

输入

- 连通无向图$G=<V,E,W>$, 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$T=<V_T,E_T>$

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

优化目标

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

问题定义

最小生成树问题

Minimum Spanning Tree Problem

输入

- 连通无向图$G=<V,E,W>$, 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$T=<V_T,E_T>$

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

优化目标

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

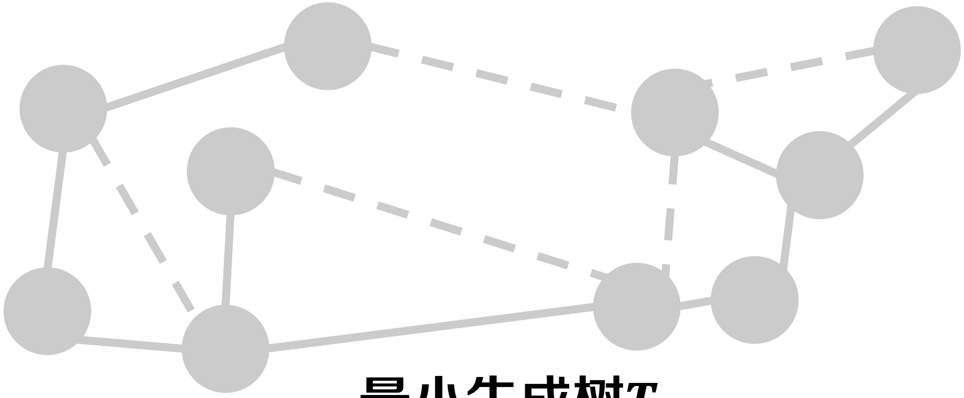
约束条件

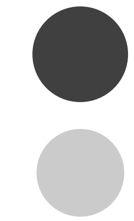
提纲

- 问题背景
- 通用框架
- Prim算法
- 算法实例
- 算法分析

通用框架

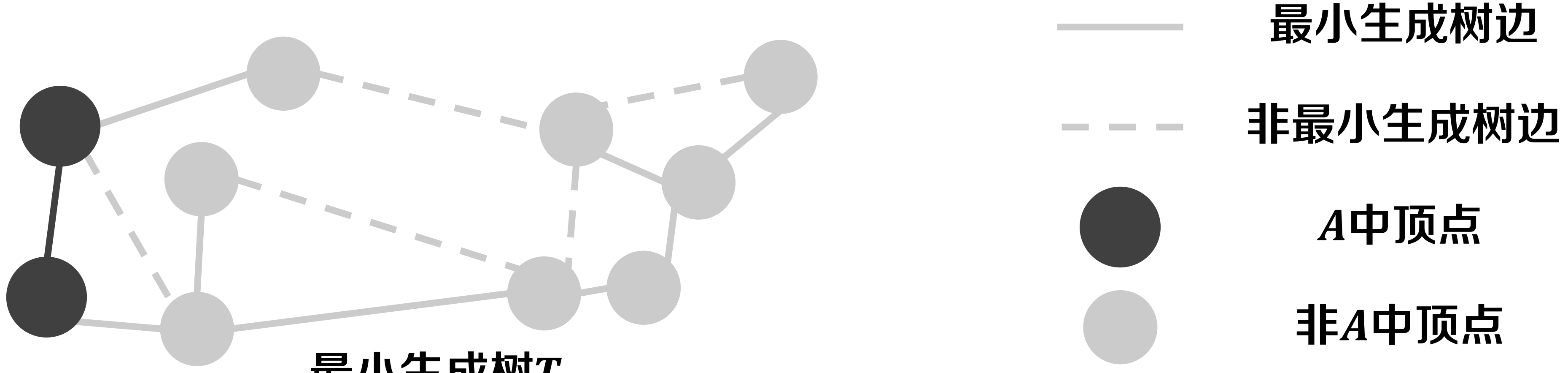
- 生成树是一个无向图中的连通、无环的生成子图
  - 新建一个空边集A，边集A可逐步扩展为最小生成树





通用框架

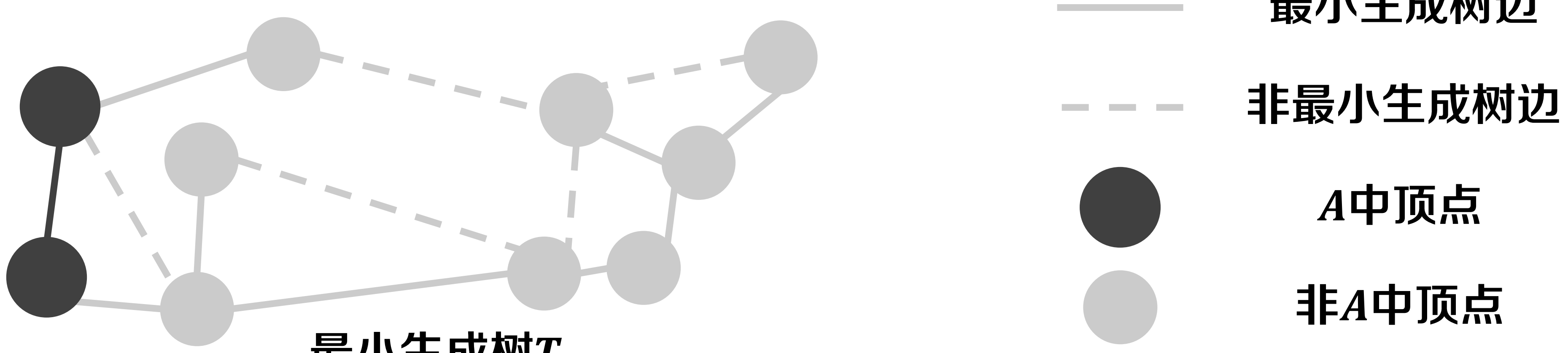
- 生成树是一个无向图中的连通、无环的生成子图
  - 新建一个空边集A，边集A可逐步扩展为最小生成树
  - 每次向边集A中新增加一条边



最小生成树T

通用框架

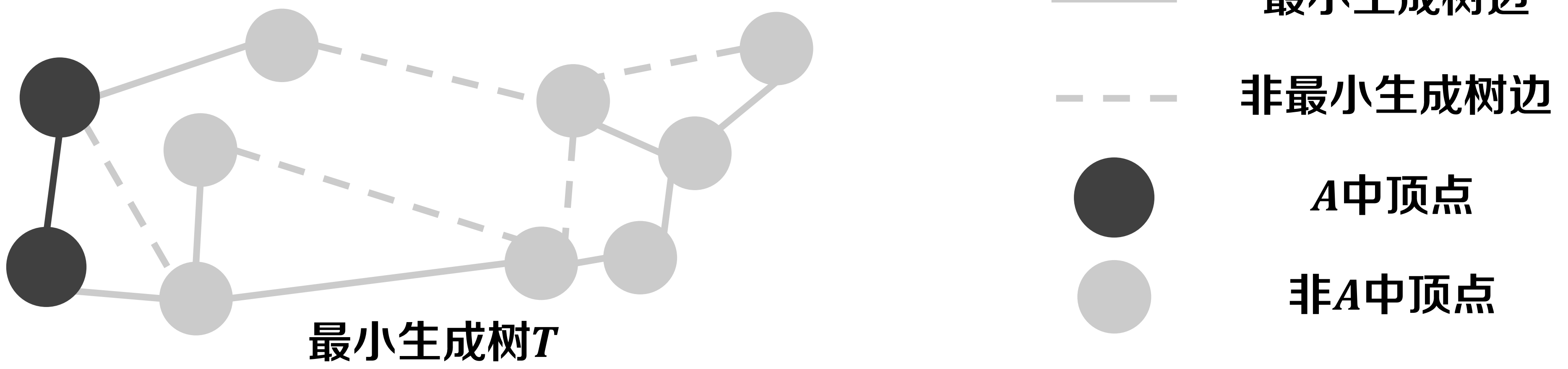
- 生成树是一个无向图中的连通、无环的生成子图
  - 新建一个空边集A，边集A可逐步扩展为最小生成树
  - 每次向边集A中新增加一条边
    - 需保证边集A仍是一个无环图
    - 需保证边集A仍是最小生成树的子集



最小生成树T

通用框架

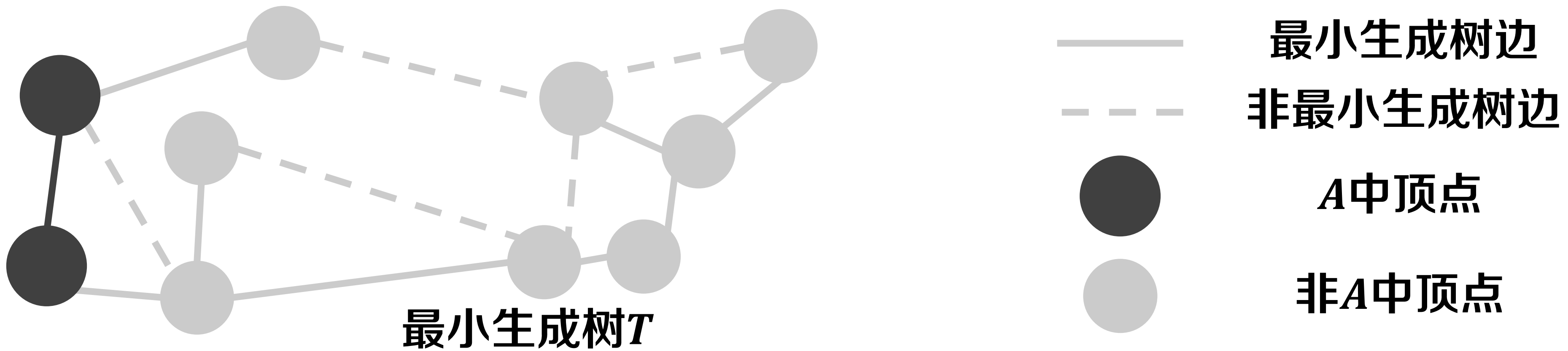
- 生成树是一个无向图中的连通、无环的生成子图
  - 新建一个空边集A，边集A可逐步扩展为最小生成树
  - 每次向边集A中新增加一条边
    - 需保证边集A仍是一个无环图
    - 需保证边集A仍是最小生成树的子集



问题：如何保证边集A仍是最小生成树的子集？

相关概念

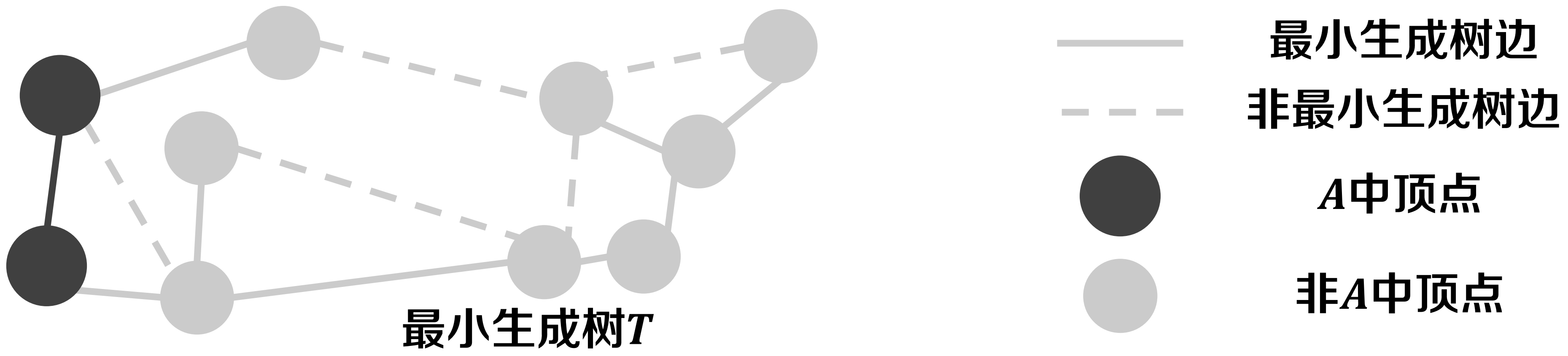
- **安全边(Safe Edge)**
  - A是某棵最小生成树T边的子集，$A \subseteq T$
  - $A \cup \{(u,v)\}$仍是T边的一个子集，则称(u,v)是A的**安全边**



最小生成树T

相关概念

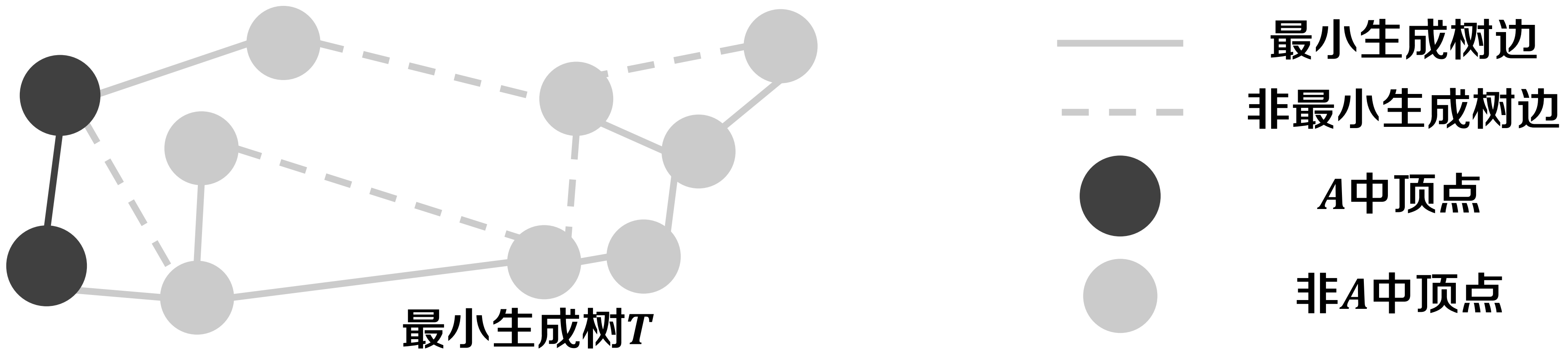
- 安全边(Safe Edge)
  - A是某棵最小生成树T边的子集，$A \subseteq T$
  - $A \cup \{(u, v)\}$仍是T边的一个子集，则称(u, v)是A的安全边



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

相关概念

- 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



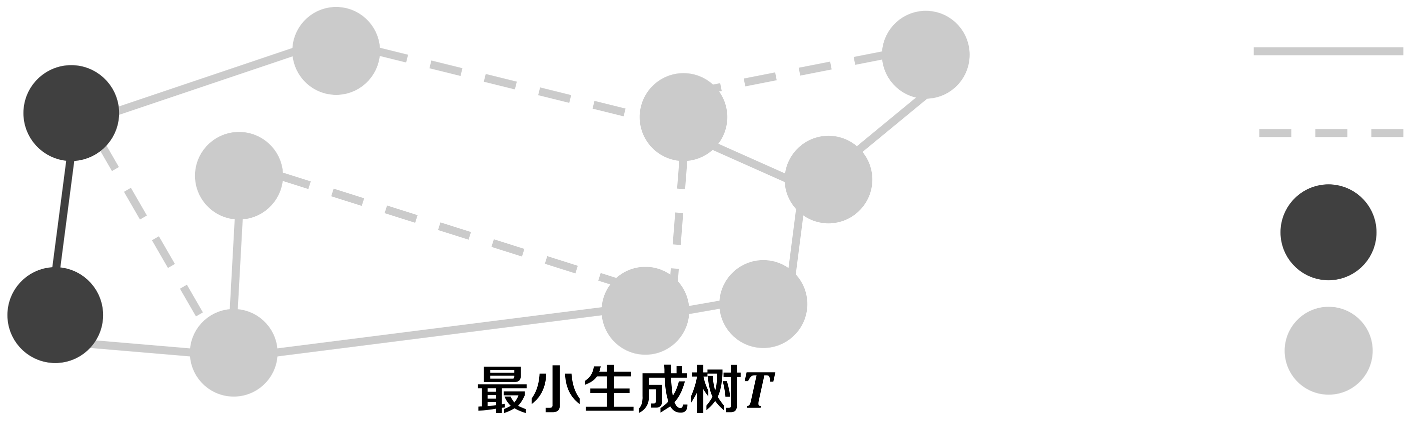
最小生成树T

- Generic-MST(G)

$A \leftarrow \emptyset$
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

相关概念

- **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**



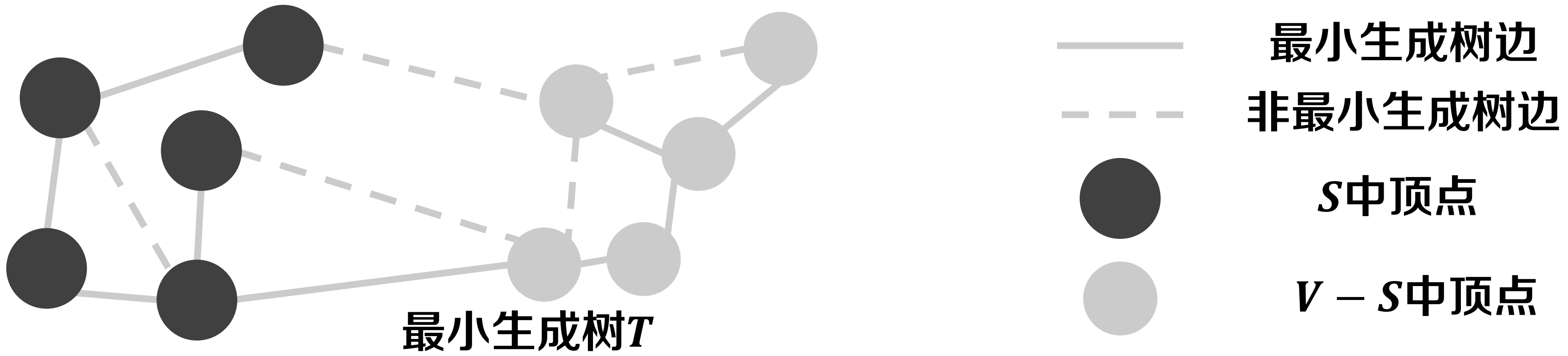
- **Generic-MST(G)**

$A \leftarrow \emptyset$
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？

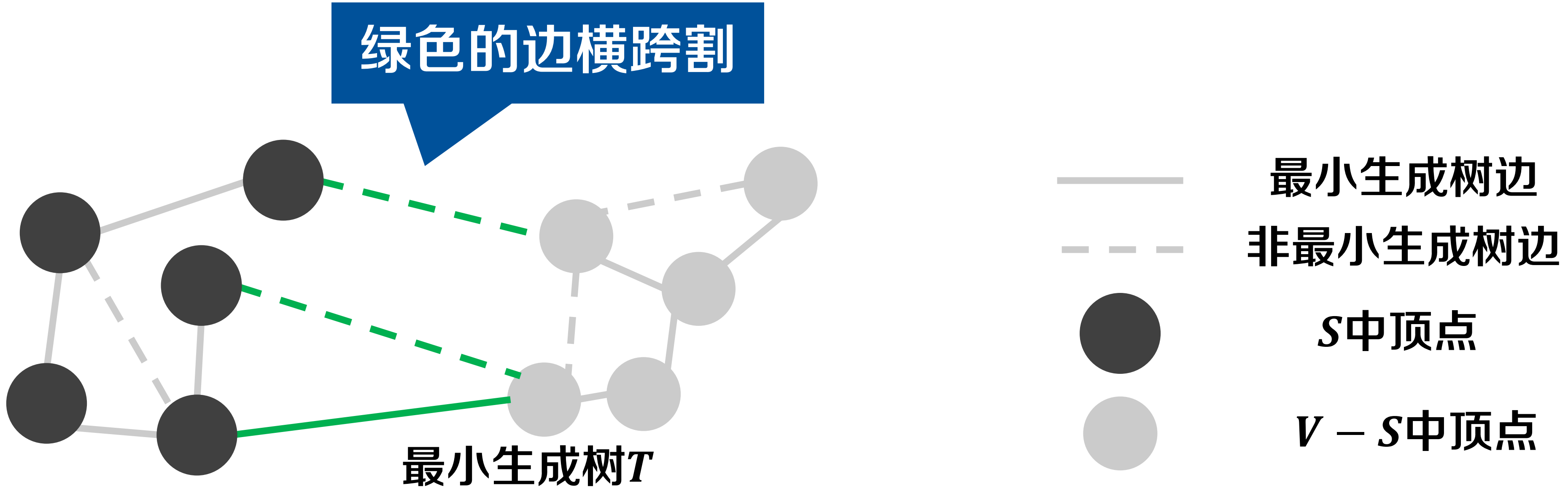
相关概念

- 割(Cut)
  - 图$G=<V,E>$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



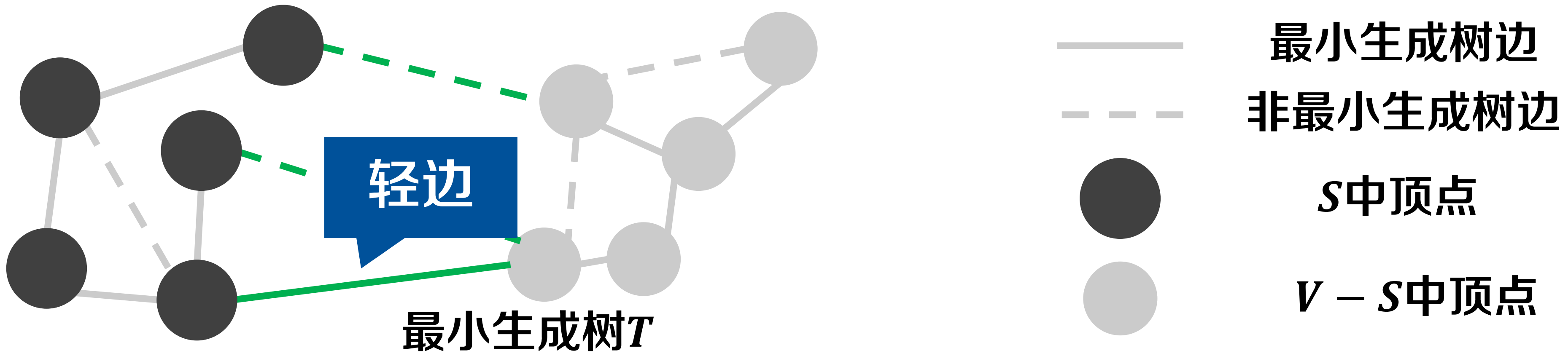
相关概念

- 割(Cut)
  - 图$G =< V, E >$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$(S, V - S)$



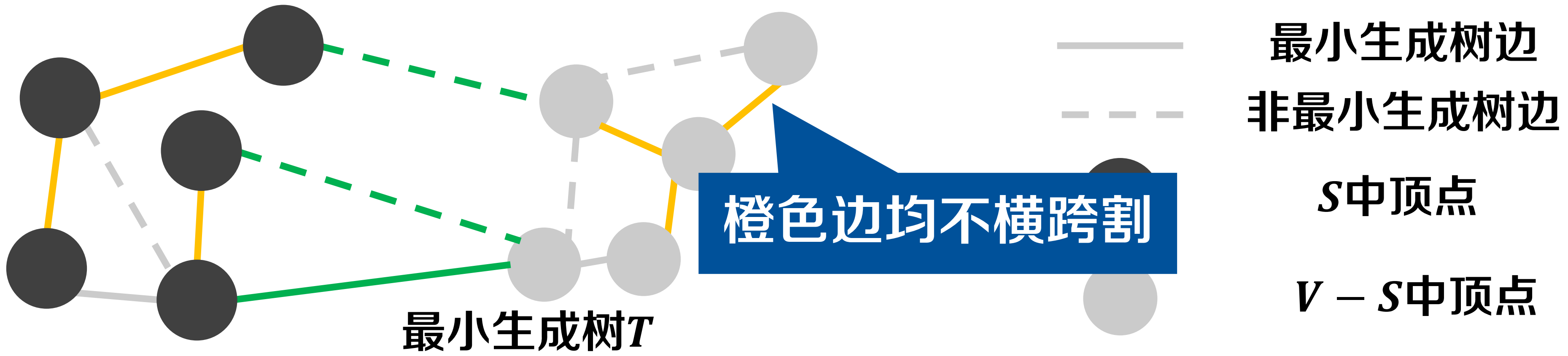
相关概念

- 割(Cut)
  - 图$G=<V,E>$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



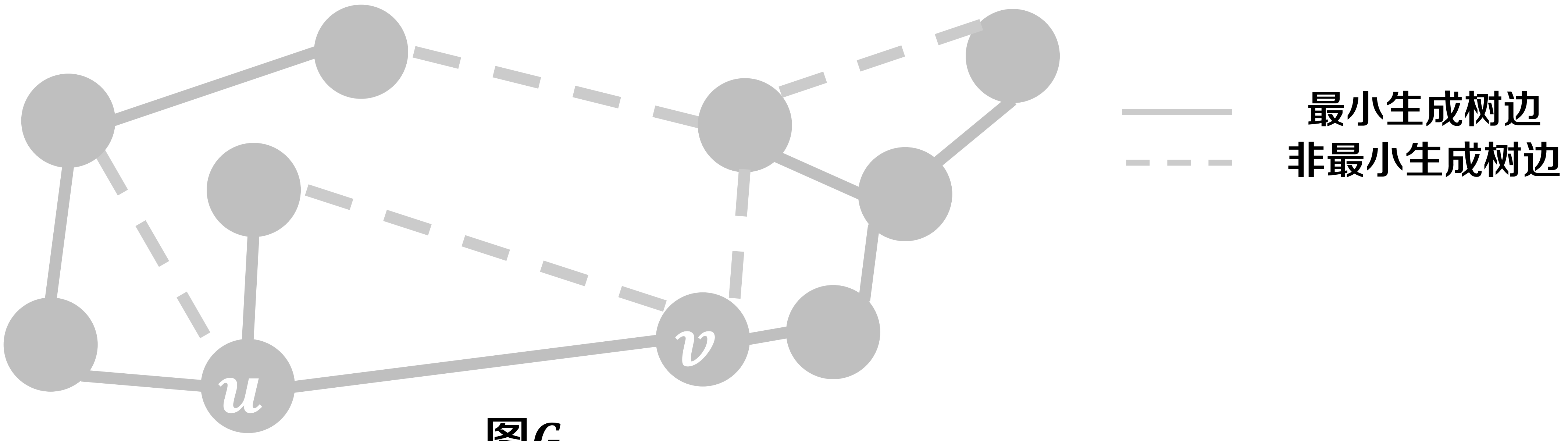
相关概念

- 割(Cut)
  - 图$G=<V,E>$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

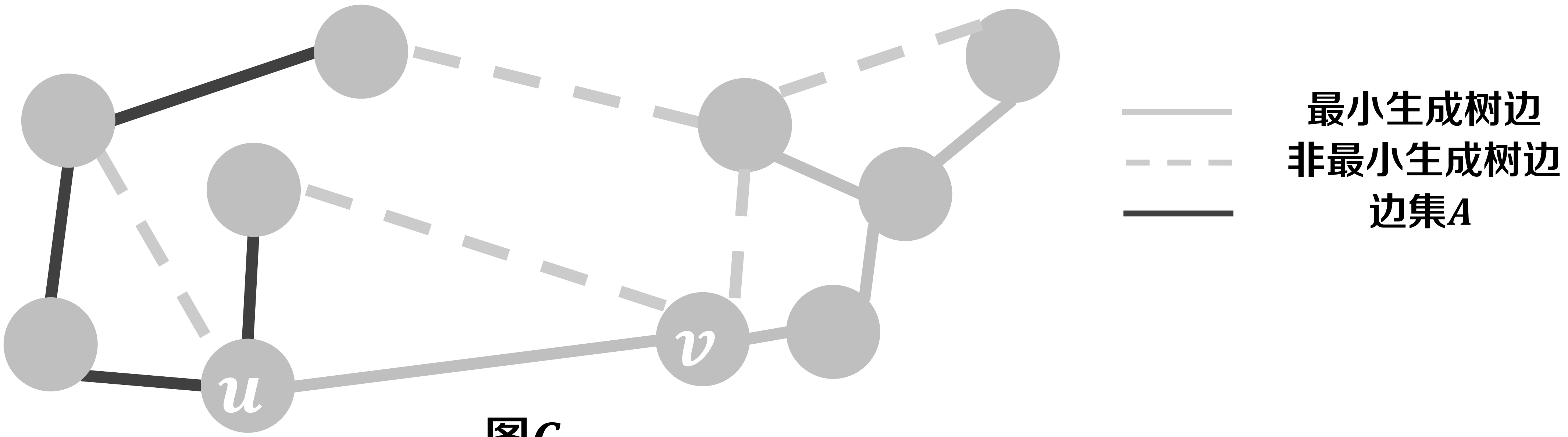
- 给定图$G=<V,E>$是一个带权的连通无向图



图G

安全边辨识定理

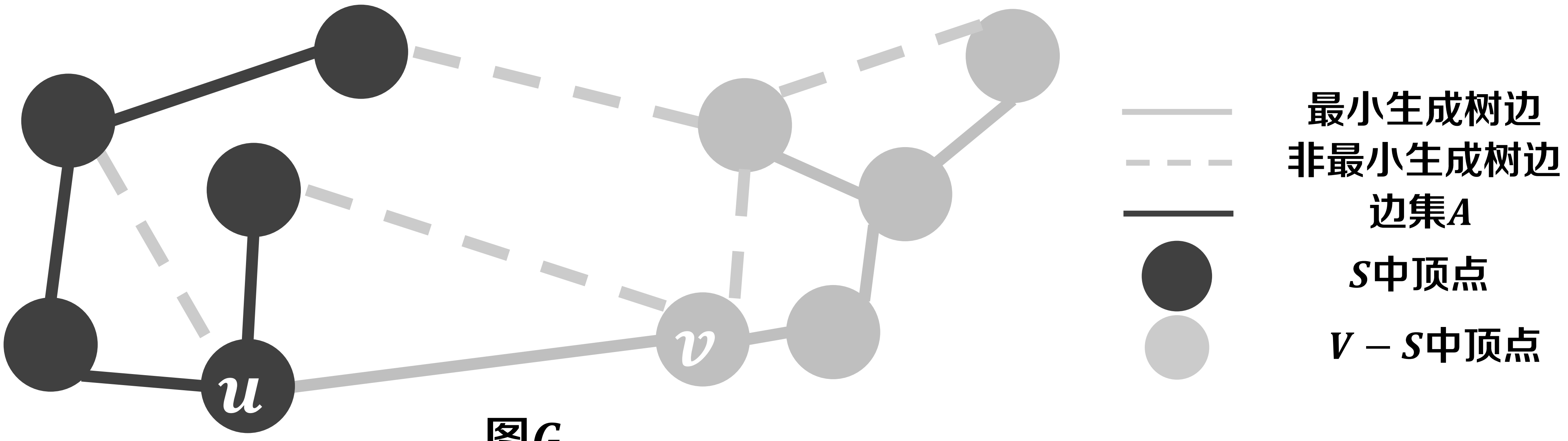
- 给定图$G=<V,E>$是一个带权的连通无向图，令A为边集E的一个子集，且A包含在图G的某棵最小生成树中



图G

安全边辨识定理

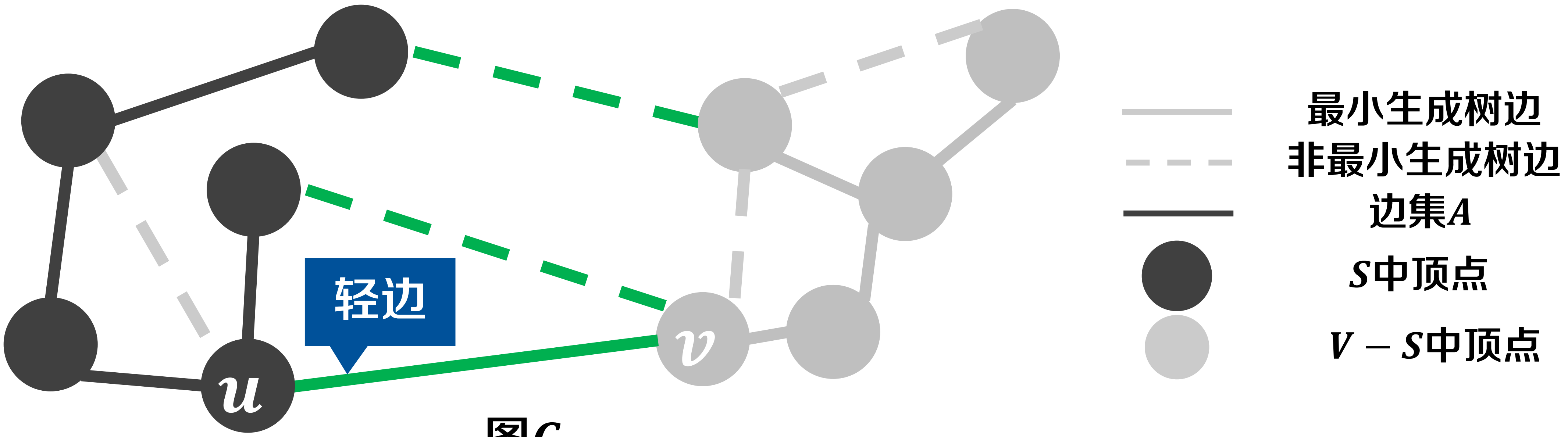
- 给定图$G=<V,E>$是一个带权的连通无向图，令A为边集E的一个子集，且A包含在图G的某棵最小生成树中
  - 若割$(S,V-S)$是图G中不妨害边集A的任意割



图G

安全边辨识定理

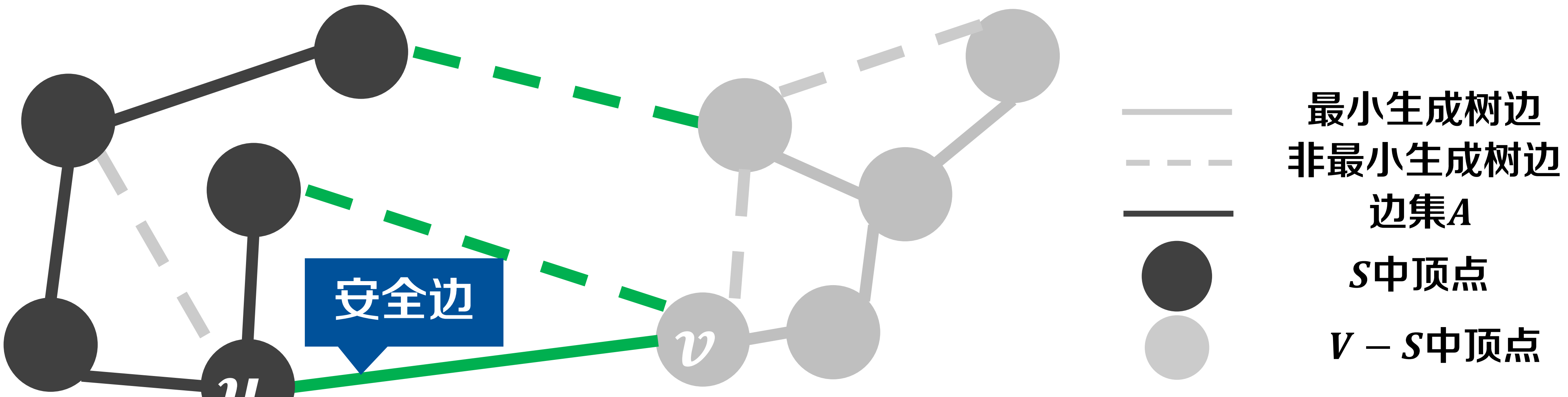
- 给定图$G=<V,E>$是一个带权的连通无向图，令A为边集E的一个子集，且A包含在图G的某棵最小生成树中
  - 若割$(S,V-S)$是图G中不妨害边集A的任意割，且(u,v)是横跨该割的轻边



图G

安全边辨识定理

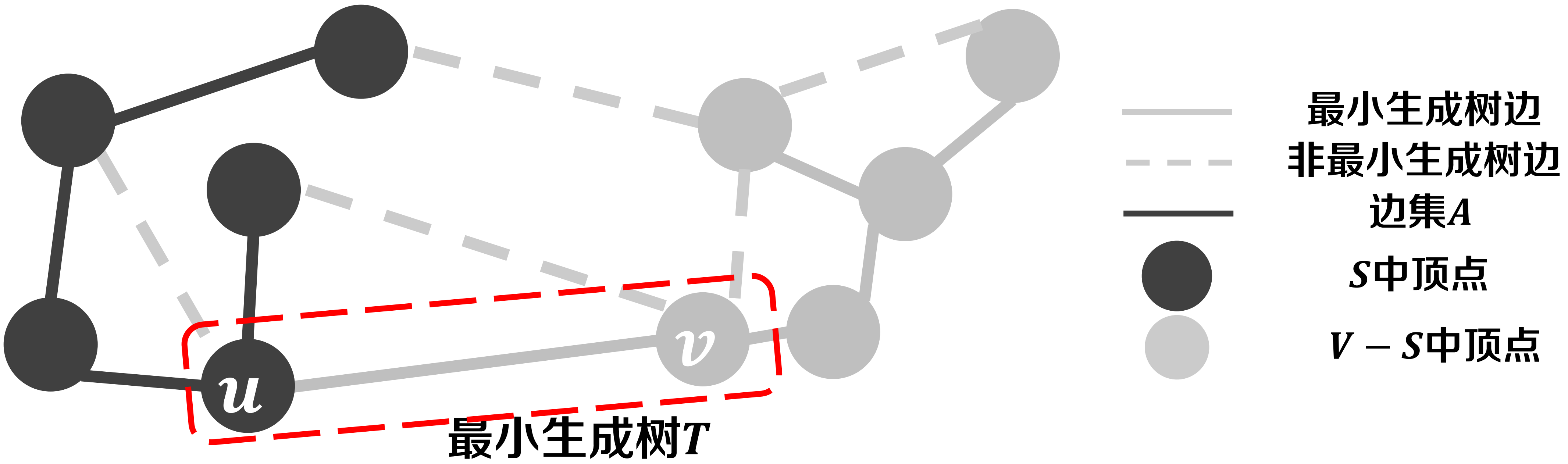
- 给定图$G=<V,E>$是一个带权的连通无向图，令A为边集E的一个子集，且A包含在图G的某棵最小生成树中
  - 若割$(S, V-S)$是图G中不妨害边集A的任意割，且(u,v)是横跨该割的轻边
  - 则对于边集A，边(u,v)是其安全边



图G

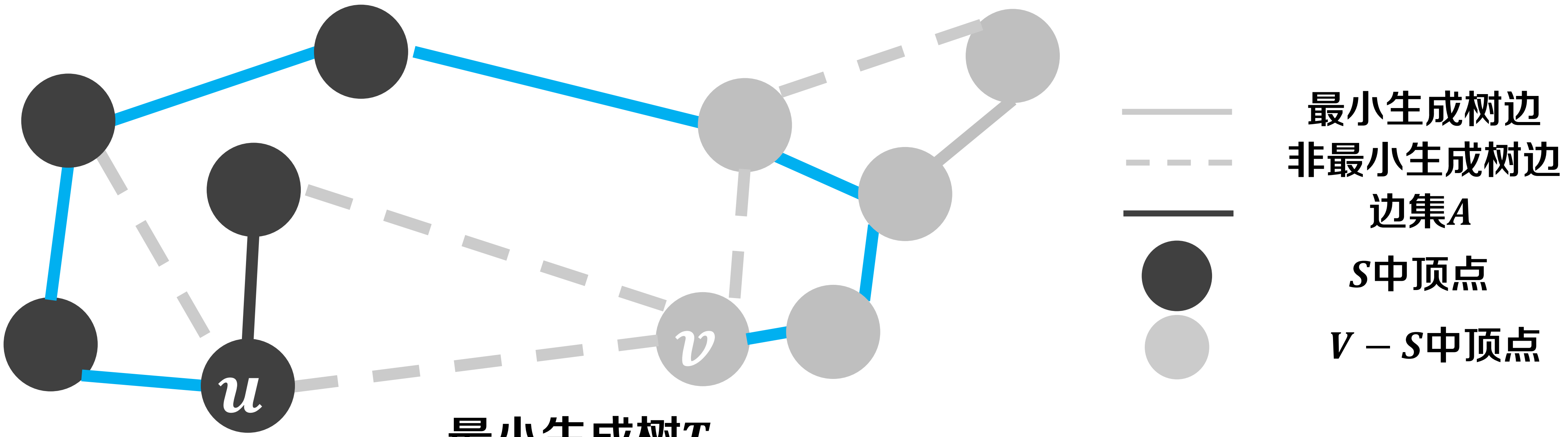
安全边辨识定理

- 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

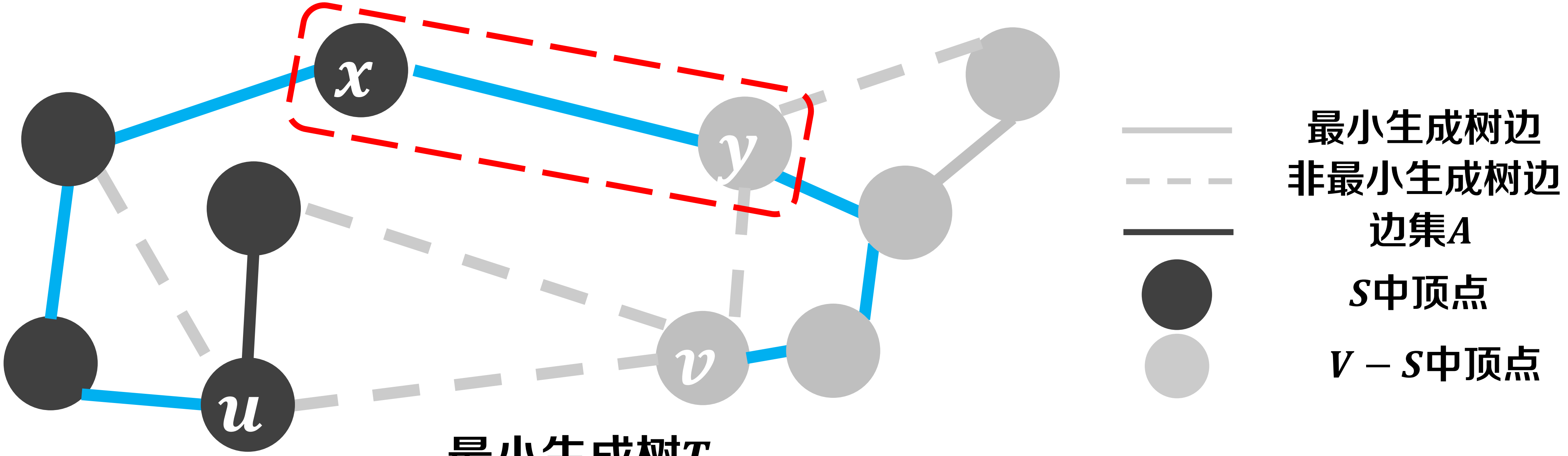
- **证明**
  - 若$(u,v)\in T$，由于$A\subseteq T$，则$A\cup\{(u,v)\}\subseteq T$，由安全边定义可证
  - 若$(u,v)\notin T$，则T中必存在u到v的路径P



最小生成树T

安全边辨识定理

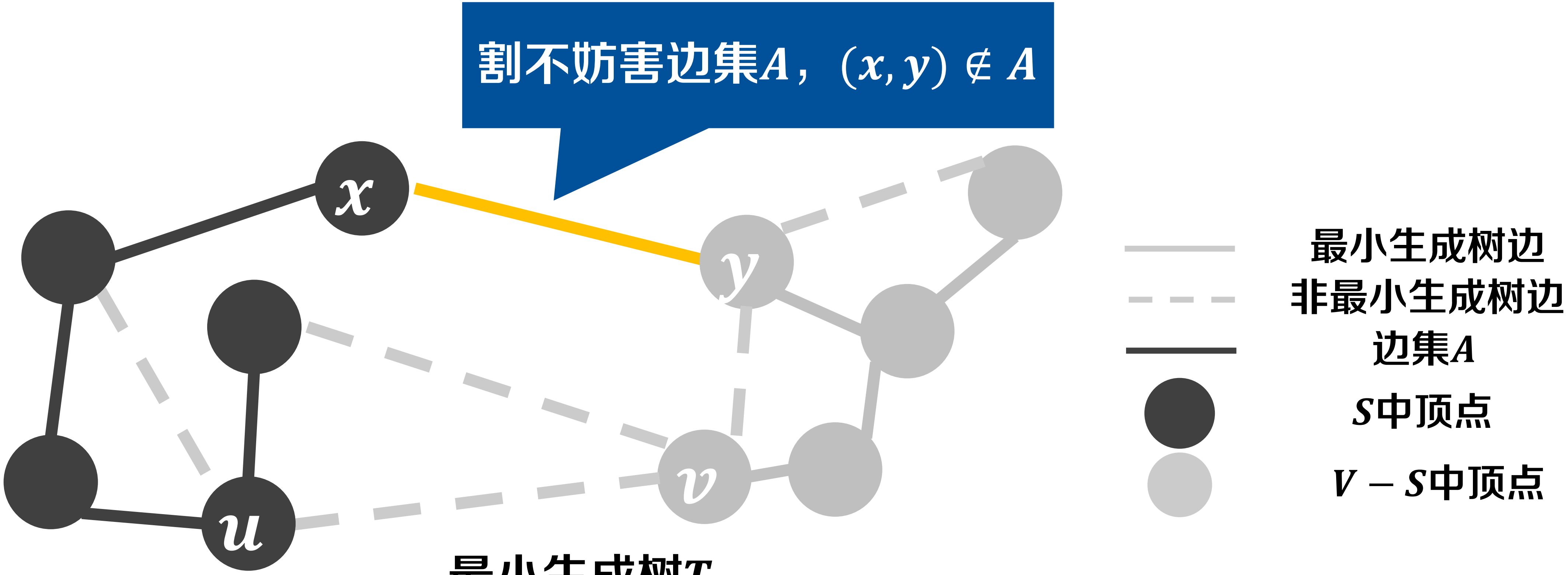
- 证明
  - 若$(\boldsymbol{u},\boldsymbol{v}) \in \boldsymbol{T}$，由于$\boldsymbol{A} \subseteq \boldsymbol{T}$，则$\boldsymbol{A} \cup \{(\boldsymbol{u},\boldsymbol{v})\} \subseteq \boldsymbol{T}$，由安全边定义可证
  - 若$(\boldsymbol{u},\boldsymbol{v}) \notin \boldsymbol{T}$，则$\boldsymbol{T}$中必存在$\boldsymbol{u}$到$\boldsymbol{v}$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



最小生成树T

安全边辨识定理

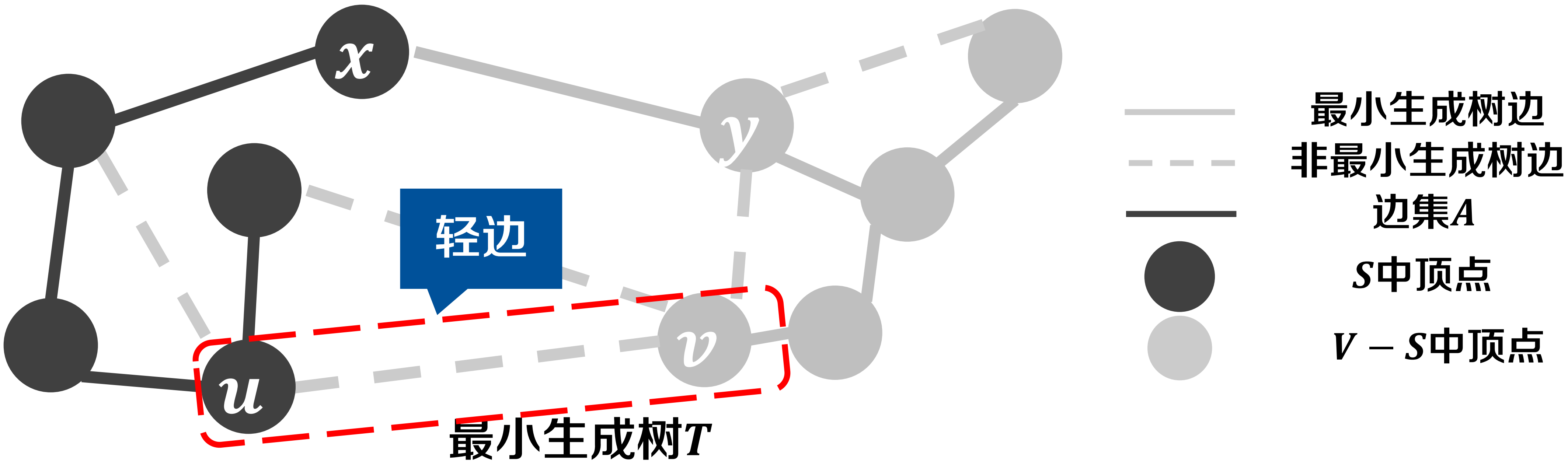
- 证明
  - 若$(u, v) \in T$，由于$A \subseteq T$，则$A \cup \{(u, v)\} \subseteq T$，由安全边定义可证
  - 若$(u, v) \notin T$，则T中必存在u到v的路径P
    - 不妨设路径P中，横跨割$(S, V-S)$的一条边为(x, y)



最小生成树T

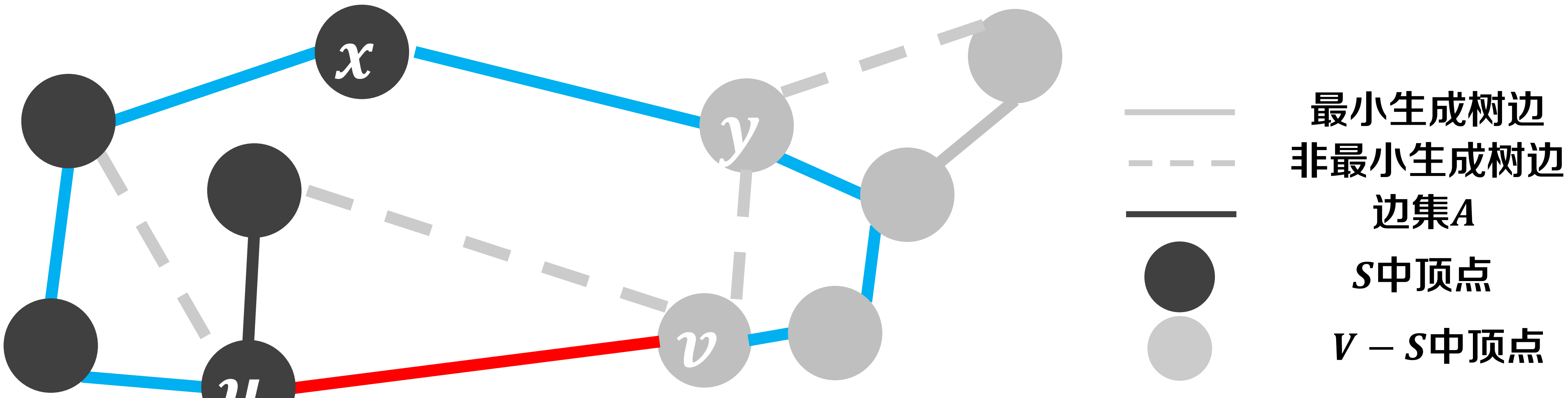
安全边辨识定理

- 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$w(u, v) \leq w(x, y)$



安全边辨识定理

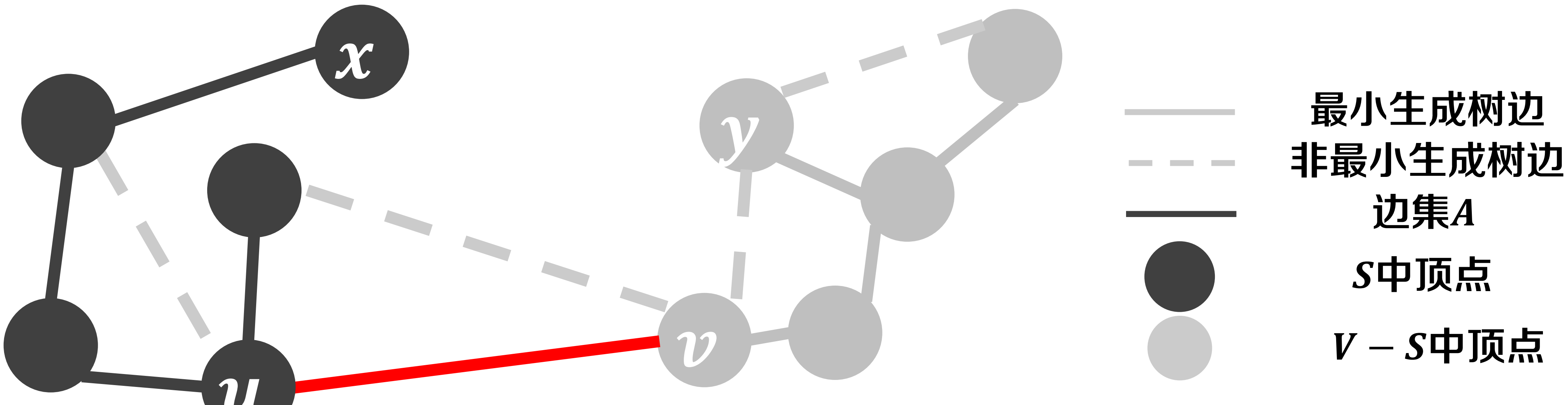
- 证明
  - 若$(u,v)\in T$，由于$A\subseteq T$，则$A\cup\{(u,v)\}\subseteq T$，由安全边定义可证
  - 若$(u,v)\notin T$，则T中必存在u到v的路径P
    - 不妨设路径P中，横跨割$(S,V-S)$的一条边为(x,y)
    - 边(u,v)是横跨割的轻边，所以$w(u,v)\le w(x,y)$
    - 将边(u,v)加入到T中会形成环路



最小生成树T

安全边辨识定理

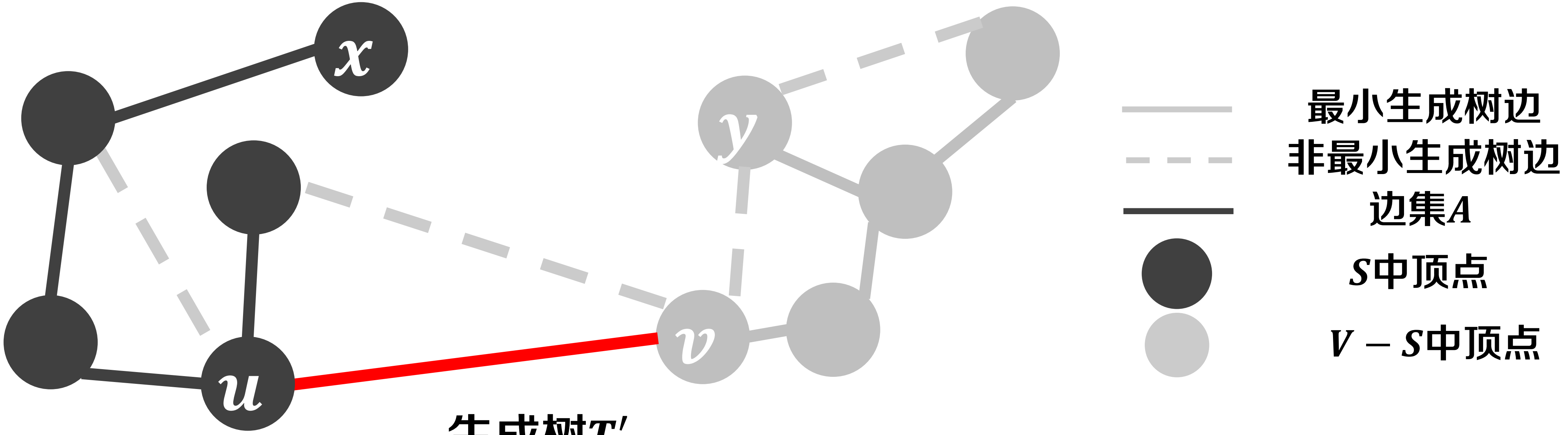
- 证明
  - 若$(u,v) \in T$，由于$A \subseteq T$，则$A \cup \{(u,v)\} \subseteq T$，由安全边定义可证
  - 若$(u,v) \notin T$，则T中必存在u到v的路径P
    - 不妨设路径P中，横跨割$(S,V-S)$的一条边为(x,y)
    - 边(u,v)是横跨割的轻边，所以$w(u,v) \leq w(x,y)$
    - 将边(u,v)加入到T中会形成环路，再去掉边(x,y)会形成另一棵树T'



生成树T'

安全边辨识定理

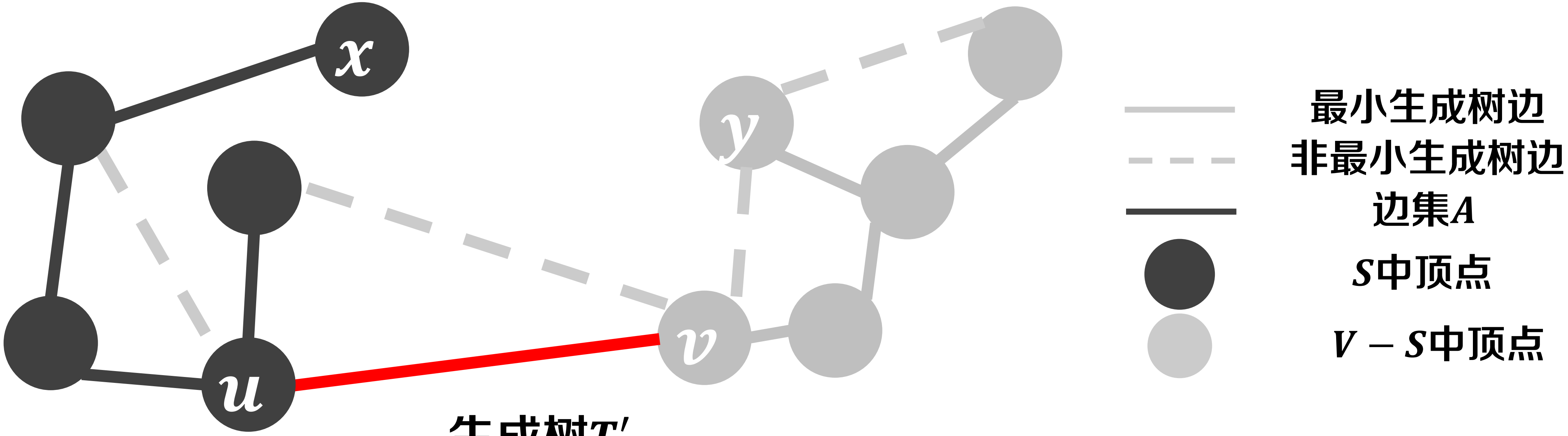
- **证明**
  - 若$(u,v) \in T$，由于$A \subseteq T$，则$A \cup \{(u,v)\} \subseteq T$，由安全边定义可证
  - 若$(u,v) \notin T$，则T中必存在u到v的路径P
    - 不妨设路径P中，横跨割$(S, V-S)$的一条边为(x,y)
    - 边(u,v)是横跨割的轻边，所以$w(u,v) \le w(x,y)$
    - 将边(u,v)加入到T中会形成环路，再去掉边(x,y)会形成另一棵树T'
    - $w(T') = w(T) + w(u,v) - w(x,y)$



生成树T'

安全边辨识定理

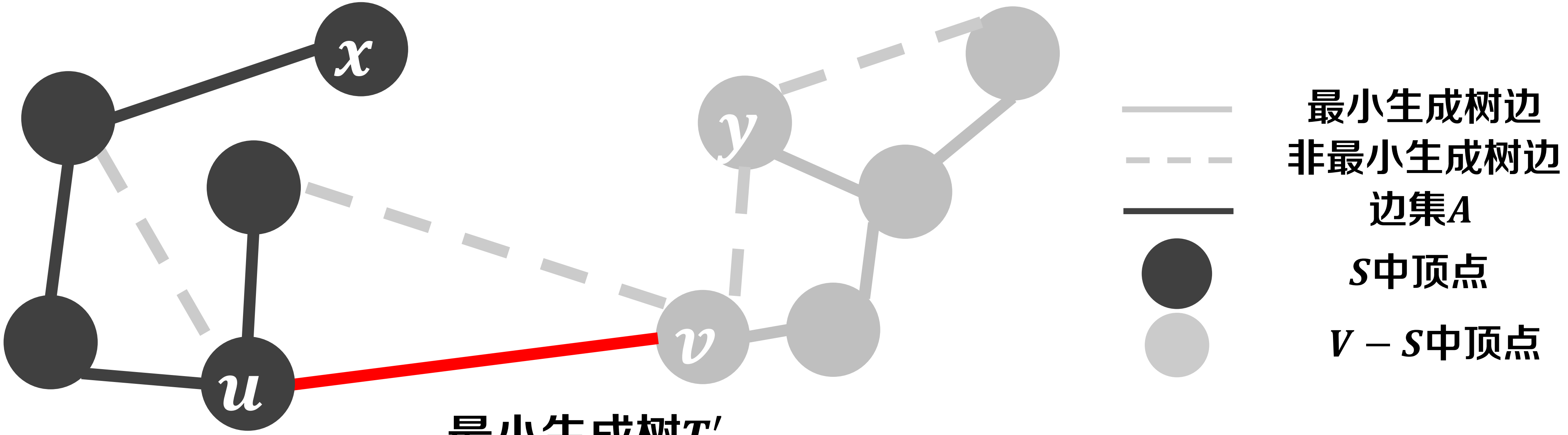
- 证明
  - 若$(u, v) \in T$，由于$A \subseteq T$，则$A \cup \{(u, v)\} \subseteq T$，由安全边定义可证
  - 若$(u, v) \notin T$，则T中必存在u到v的路径P
    - 不妨设路径P中，横跨割$(S, V-S)$的一条边为(x, y)
    - 边(u, v)是横跨割的轻边，所以$w(u, v) \leq w(x, y)$
    - 将边(u, v)加入到T中会形成环路，再去掉边(x, y)会形成另一棵树T'
    - $w(T') = w(T) + w(u, v) - w(x, y) \leq w(T)$



生成树T'

安全边辨识定理

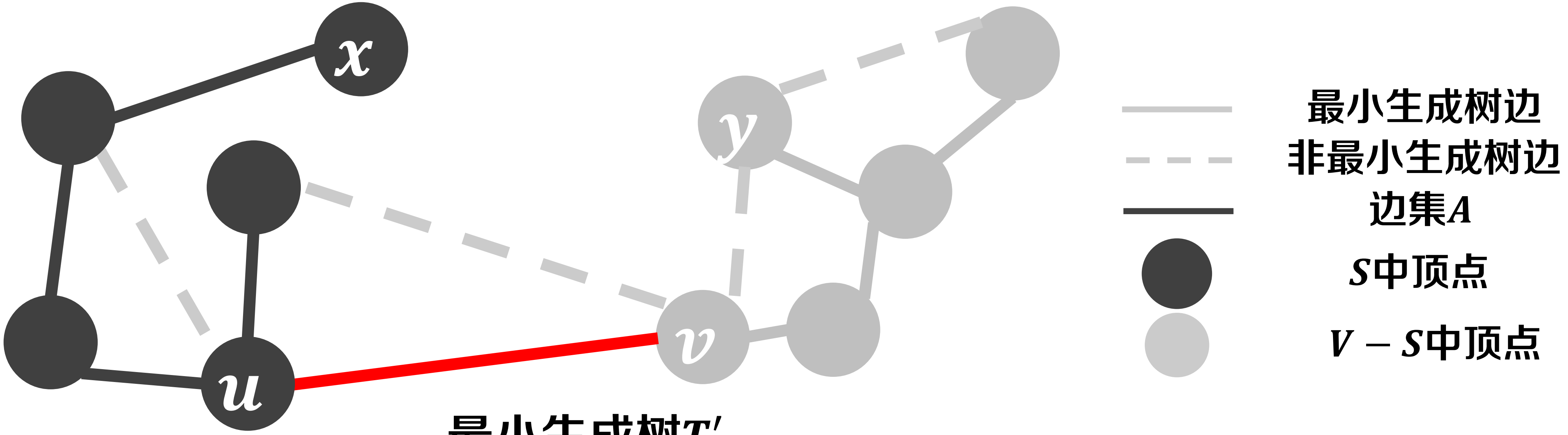
- **证明**
  - 若$(u,v)\in T$，由于$A\subseteq T$，则$A\cup\{(u,v)\}\subseteq T$，由安全边定义可证
  - 若$(u,v)\notin T$，则T中必存在u到v的路径P
    - 不妨设路径P中，横跨割$(S,V-S)$的一条边为(x,y)
    - 边(u,v)是横跨割的轻边，所以$w(u,v)\leq w(x,y)$
    - 将边(u,v)加入到T中会形成环路，再去掉边(x,y)会形成另一棵树T'
    - $w(T')\leq w(T)$，T'也是最小生成树



最小生成树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$w(u,v) \le w(x,y)$
    - 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'
    - $w(T') \le w(T)$，T'也是最小生成树，$A \cup \{(u,v)\} \subseteq T'$，边$(u,v)$是安全边



最小生成树T'

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

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

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?

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

提纲

问题背景

通用框架

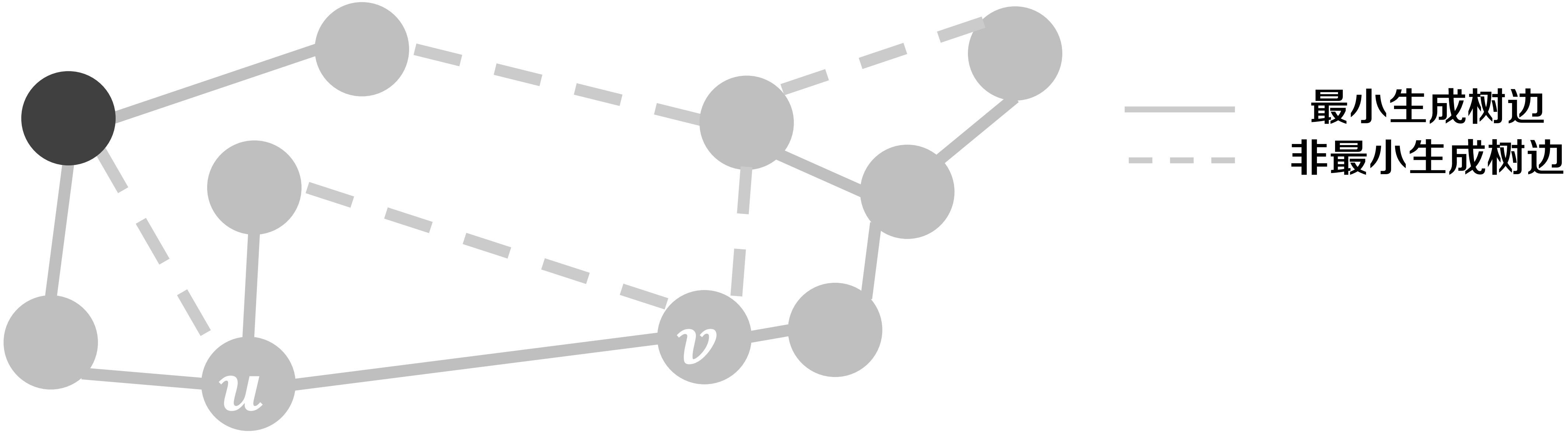
Prim算法

算法实例

算法分析

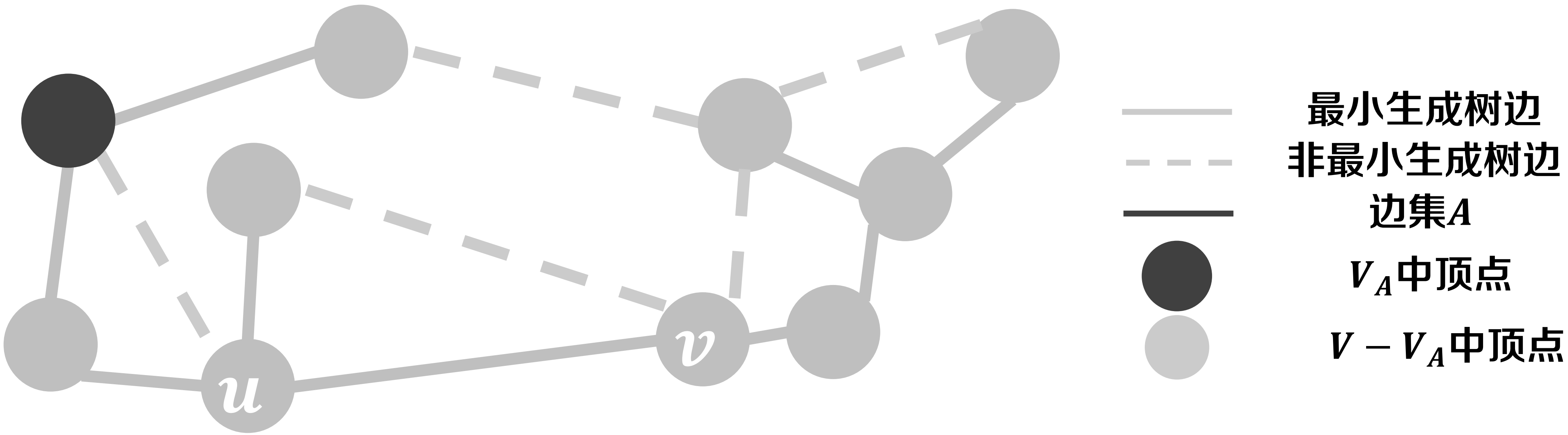
Prim算法

- 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



Prim算法

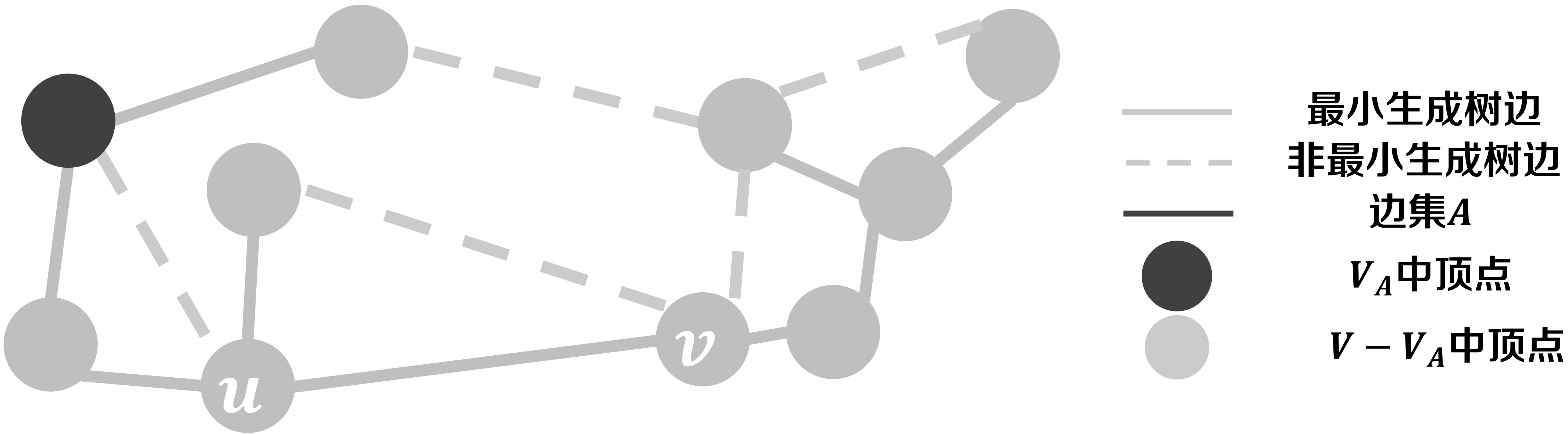
- 算法思想
  - 步骤1：选择任意一个顶点，作为生成树的起始顶点
  - 步骤2：保持边集A始终为一棵树



Prim算法

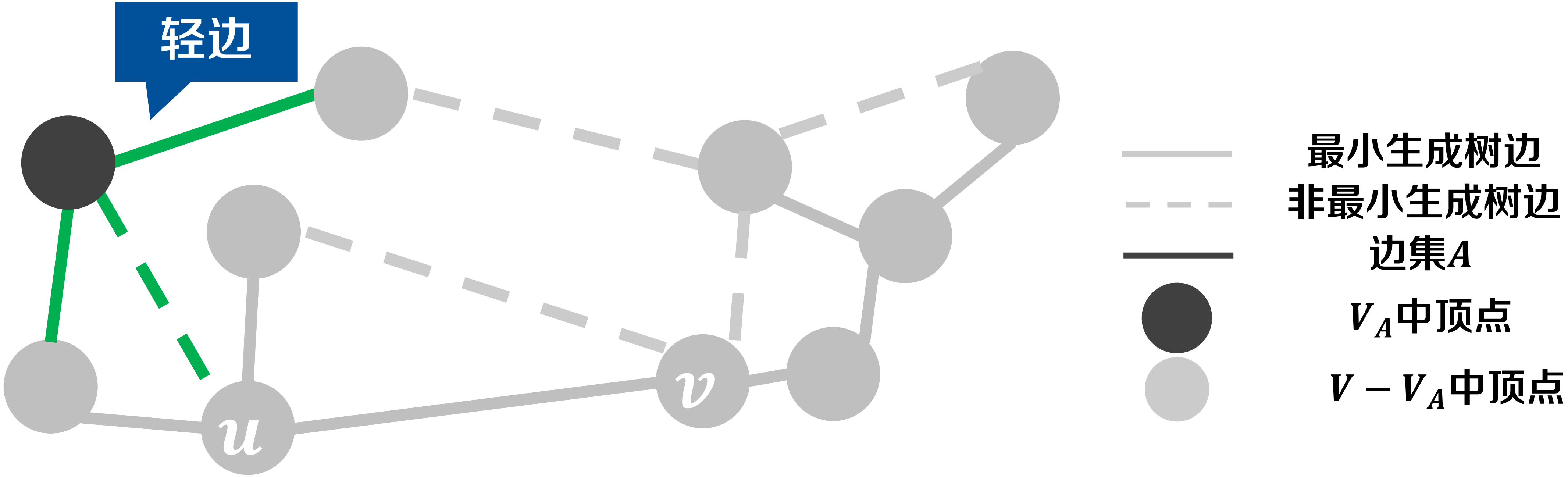
- 算法思想
  - 步骤1：选择任意一个顶点，作为生成树的起始顶点
  - 步骤2：保持边集A始终为一棵树，选择割$(V_A, V - V_A)$

树中顶点



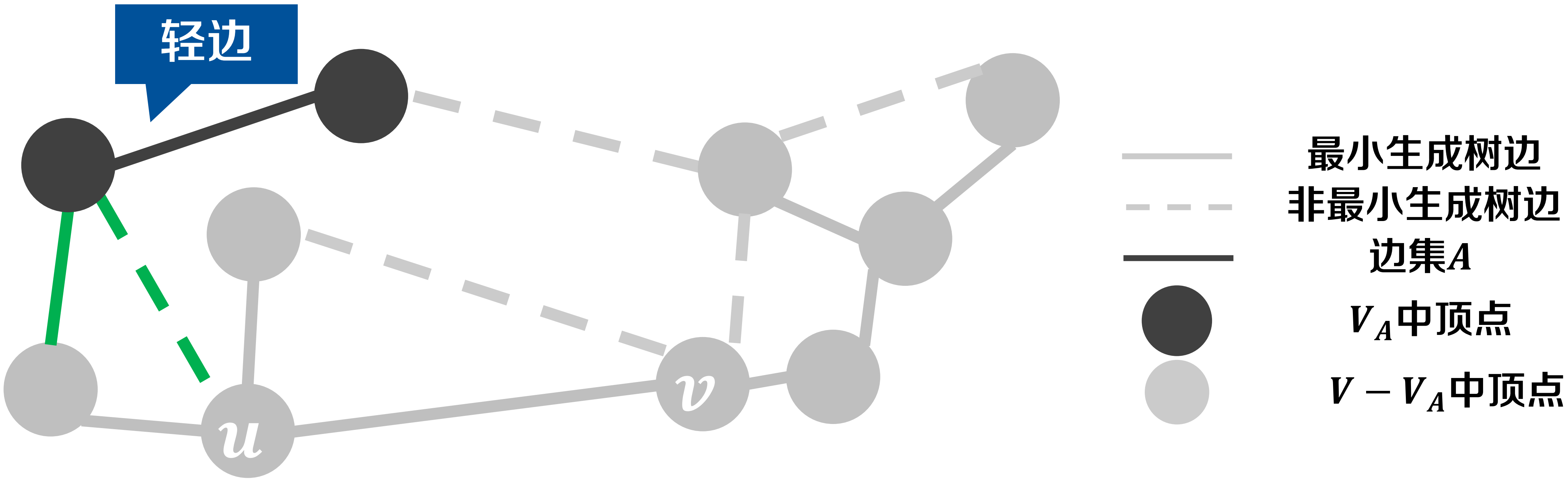
Prim算法

- 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



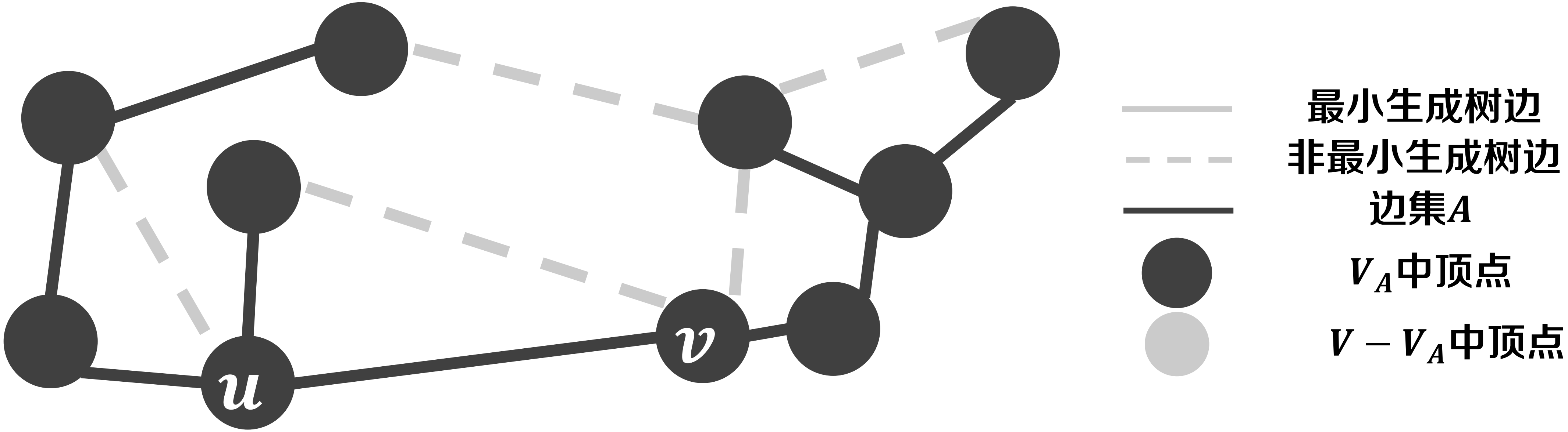
Prim算法

- 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



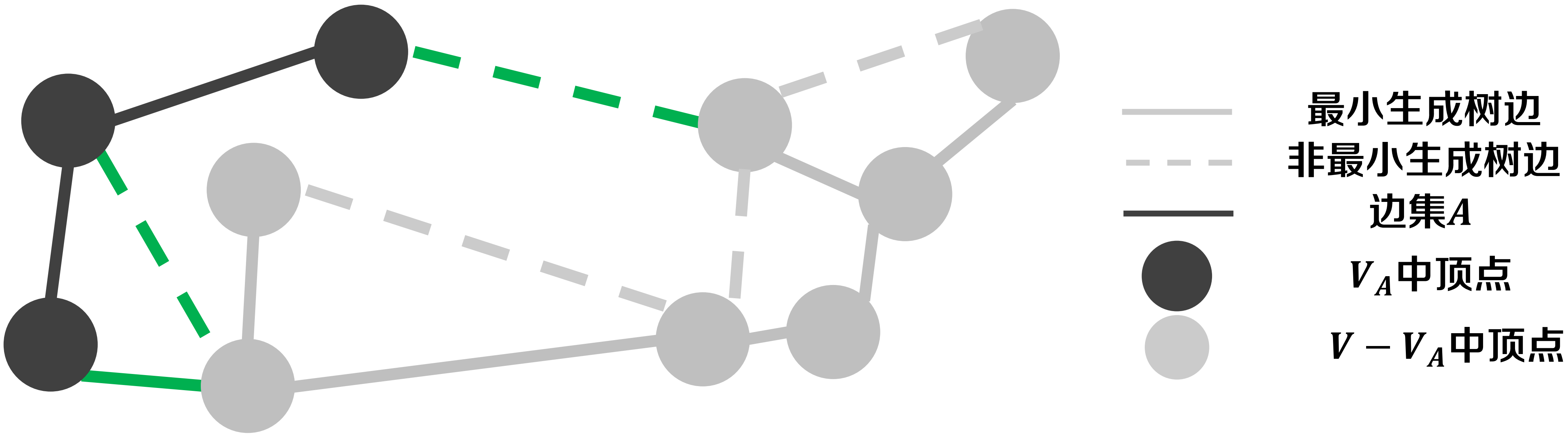
Prim算法

- 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



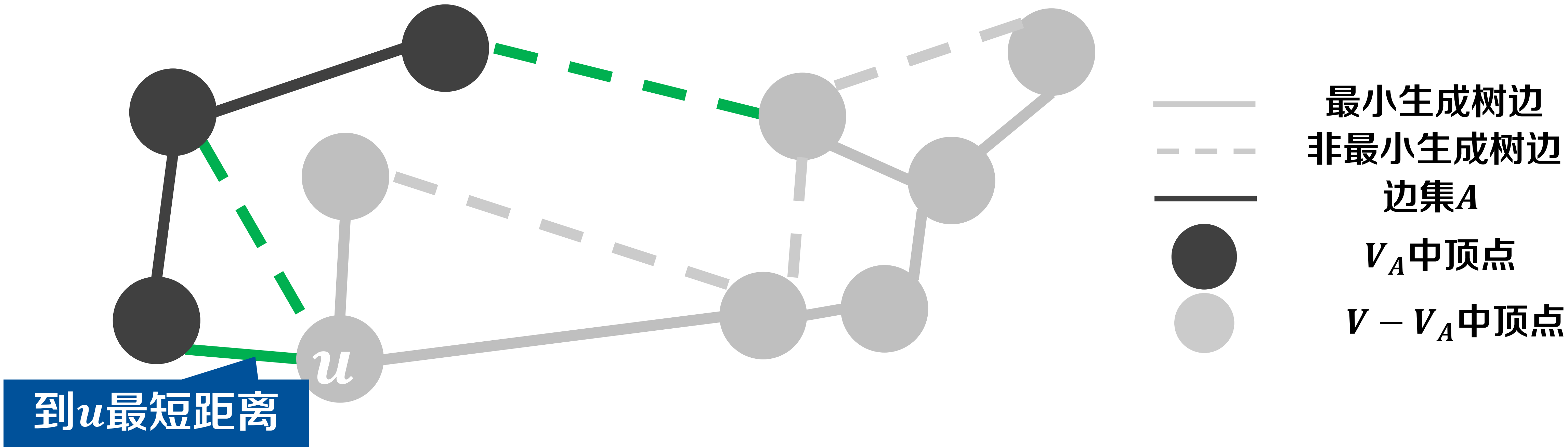
Prim算法

- 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， $u \in V - V_A$



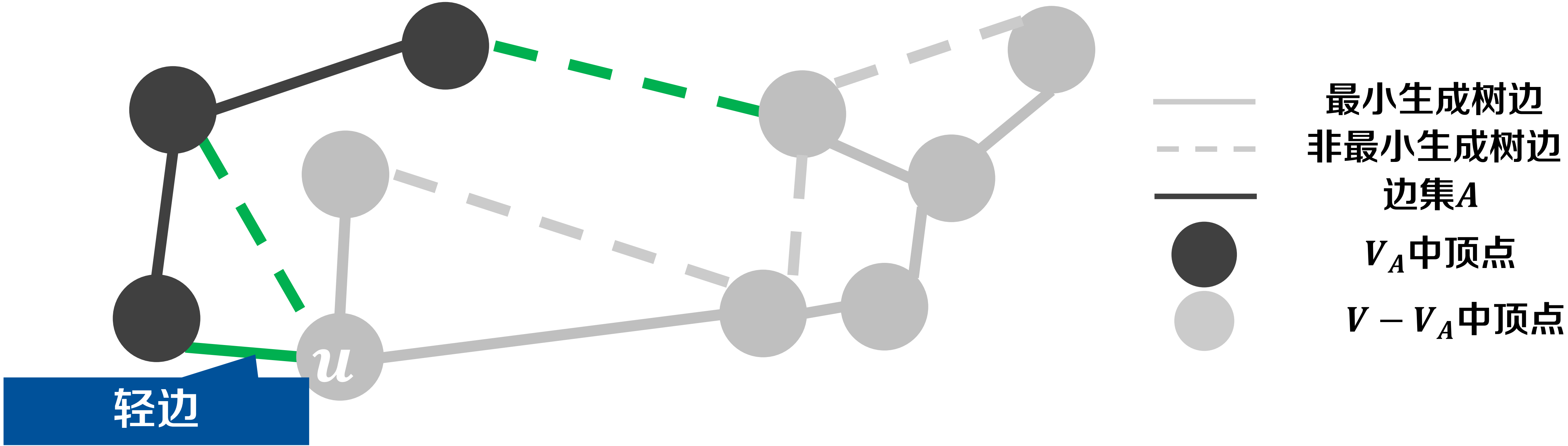
Prim算法

- 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，$dist[u] = \min\{w(x,u)\}, \forall x \in V_A$



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：$\min\{dist[u]\}, \forall u \in V - V_A$



Prim算法

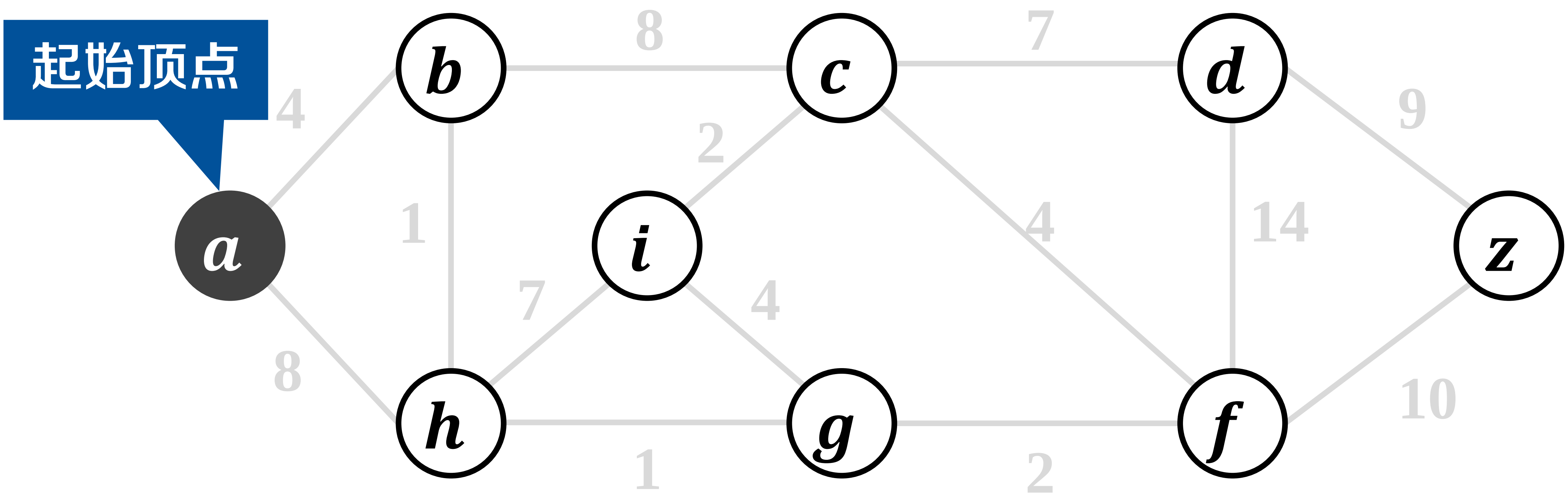
- 辅助数组
  - $color$表示顶点状态
    - 黑色顶点u已覆盖， $u \in V_A$
    - 白色顶点u未覆盖， $u \in V - V_A$
  - $dist$记录横跨$(V_A, V - V_A)$边的权重
    - 顶点集V_A到顶点u的最短距离，$dist[u] = \min\{w(x, u)\}, \forall x \in V_A$
    - 轻边： $\min\{dist[u]\}, \forall u \in V - V_A$
  - $pred$表示前驱顶点
    - $(pred[u], u)$为最小生成树的边

提纲

- 问题背景
- 通用框架
- Prim算法
- 算法实例
- 算法分析

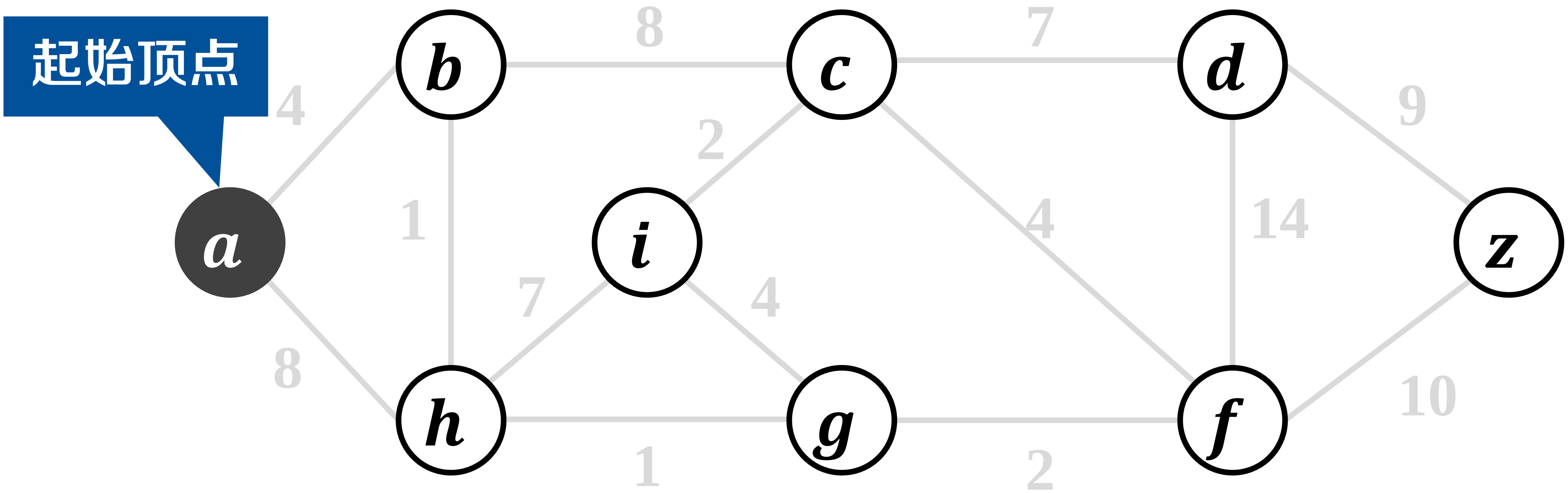
算法实例

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



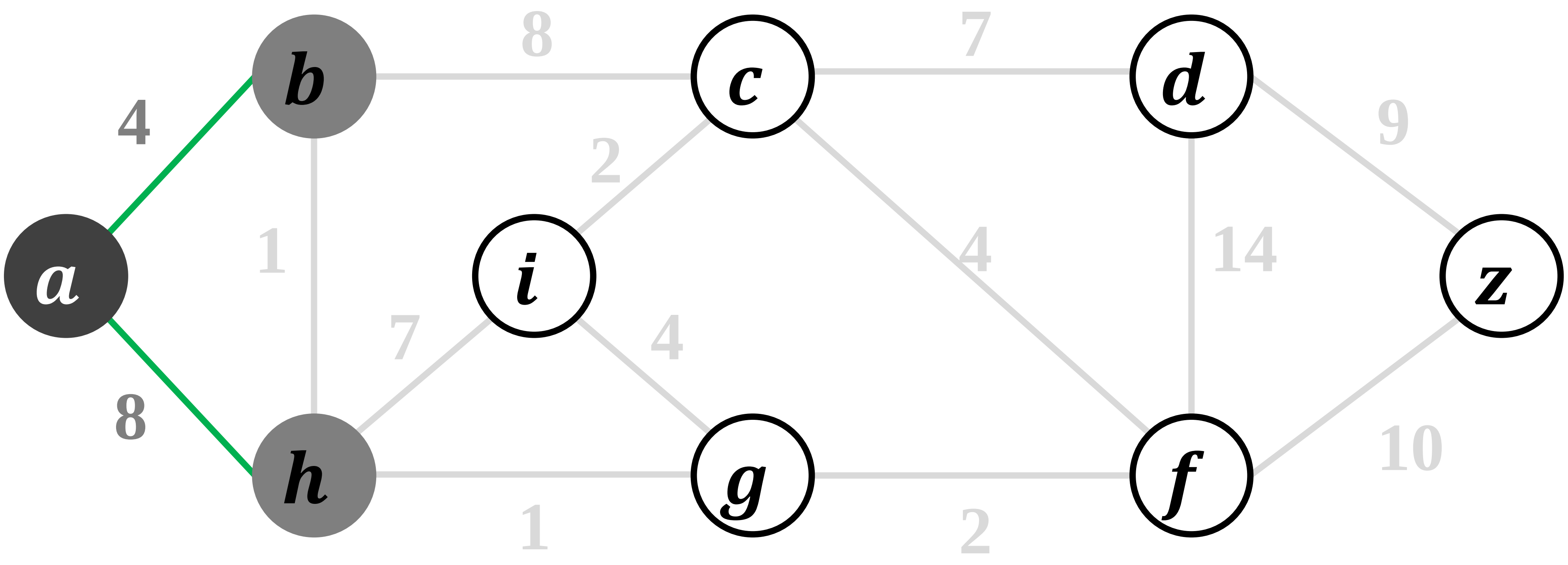
算法实例

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



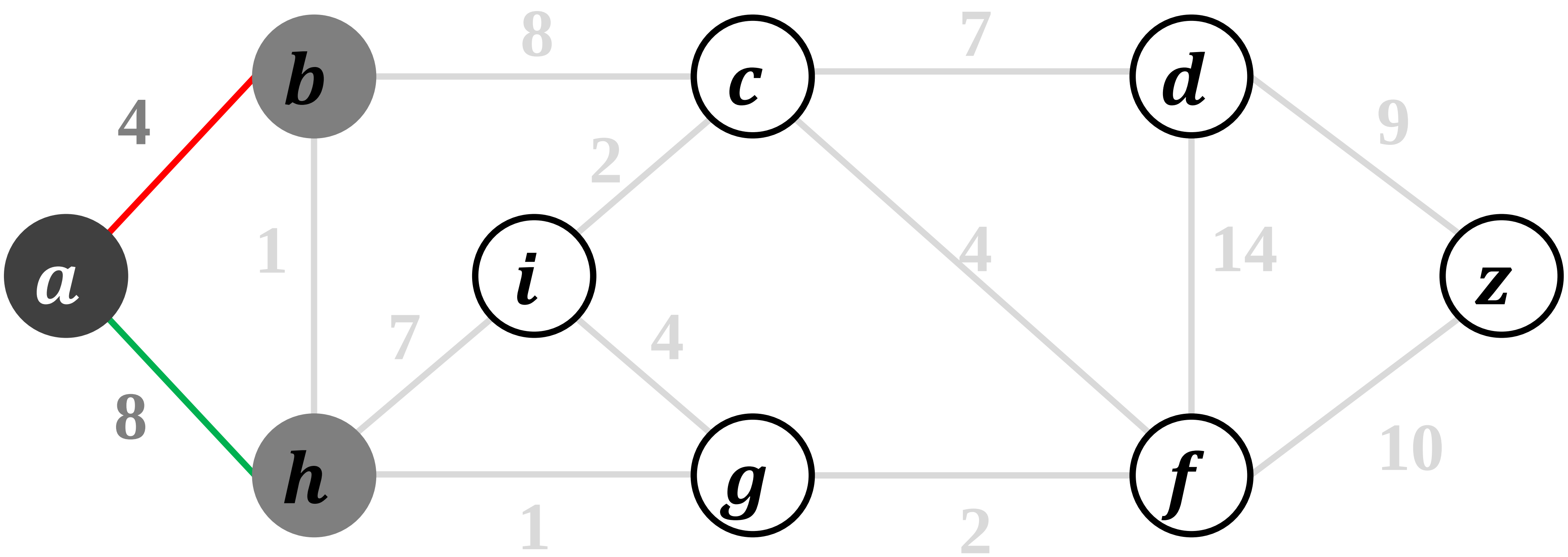
算法实例

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



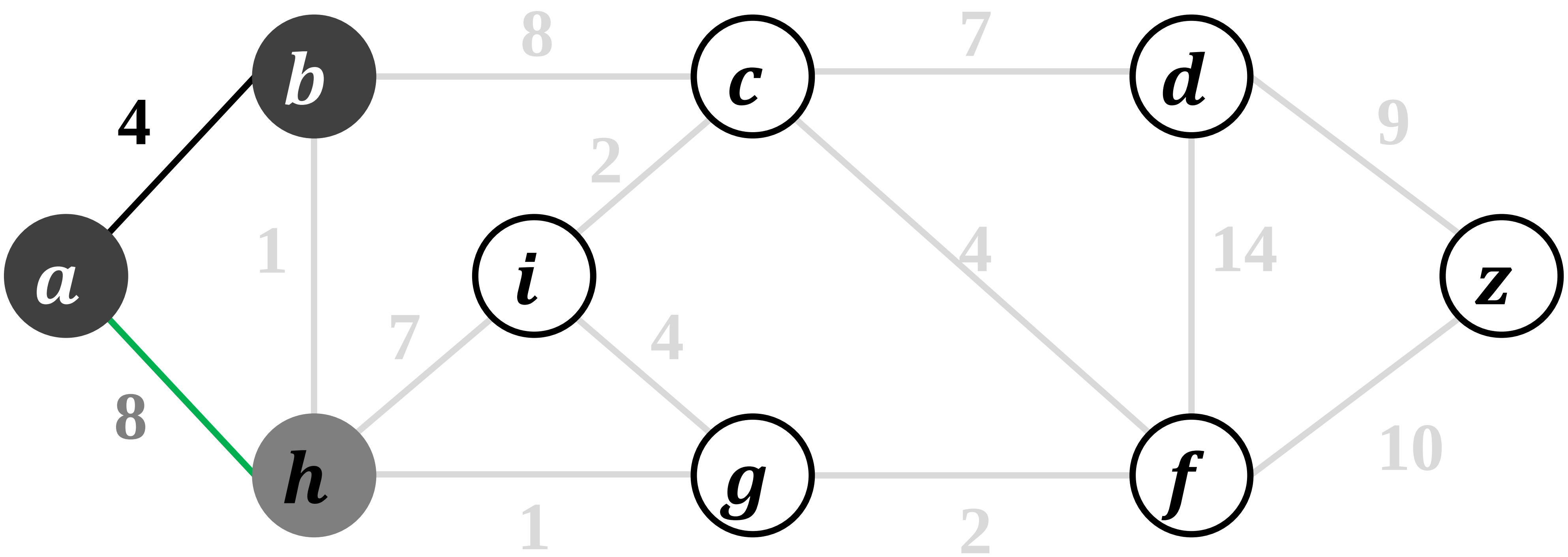
算法实例

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



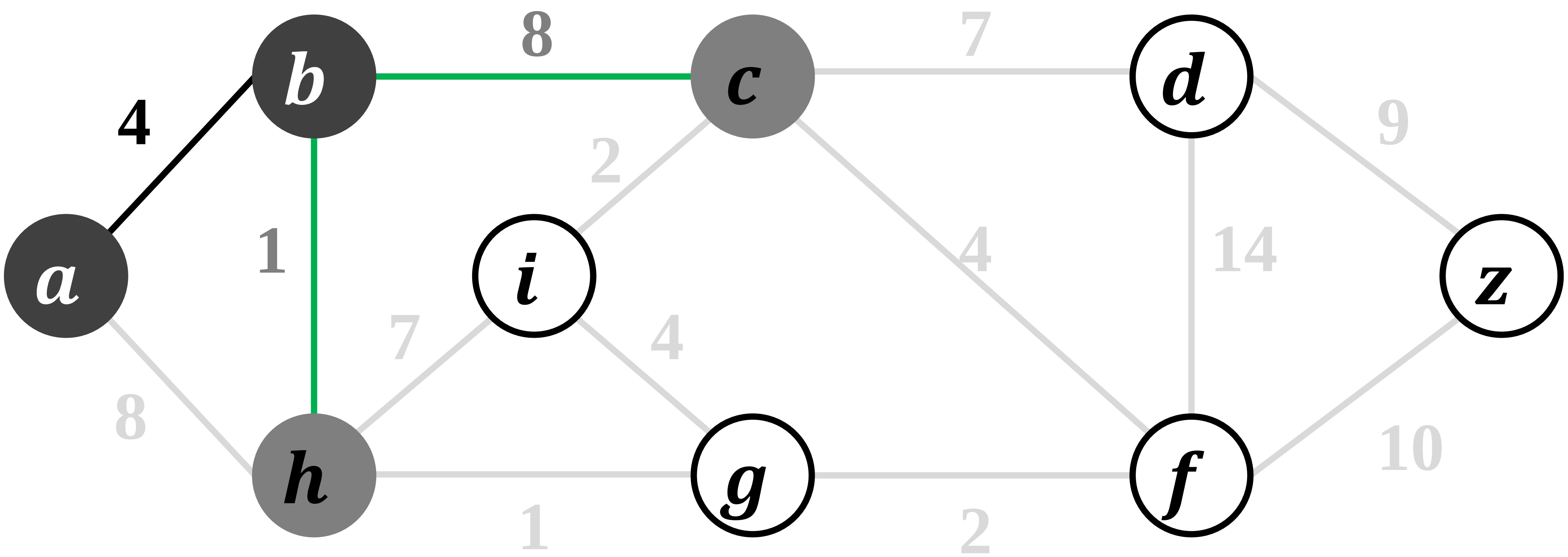
算法实例

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



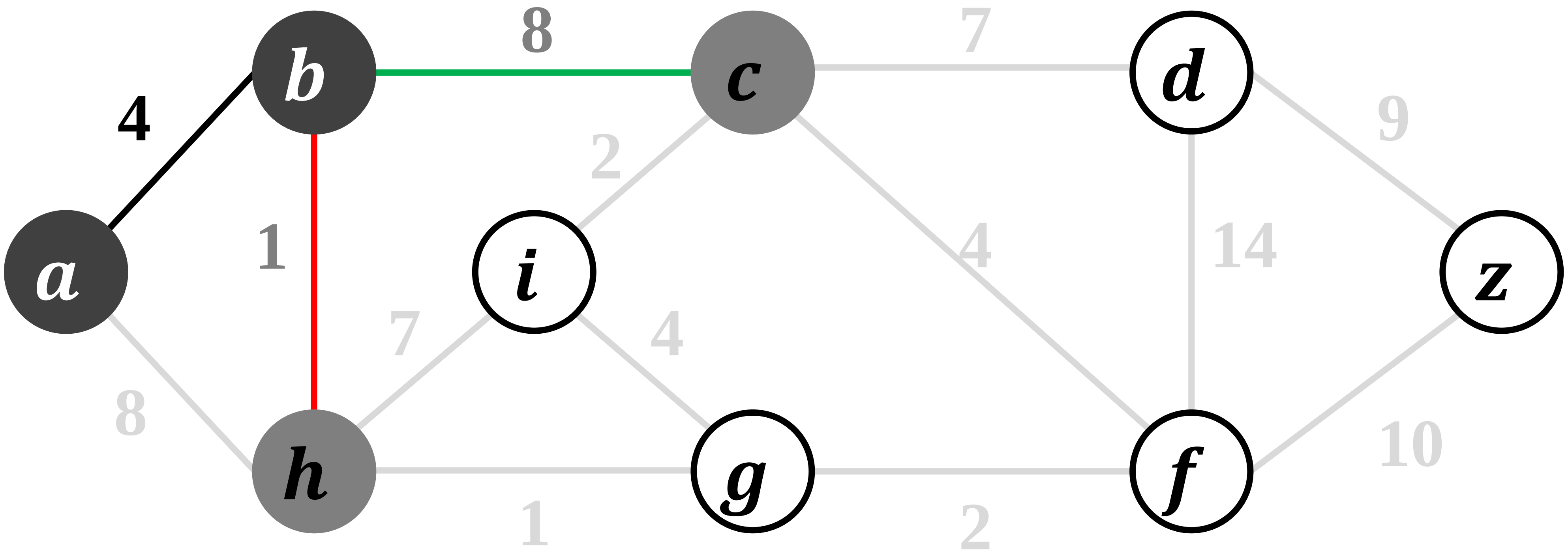
算法实例

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



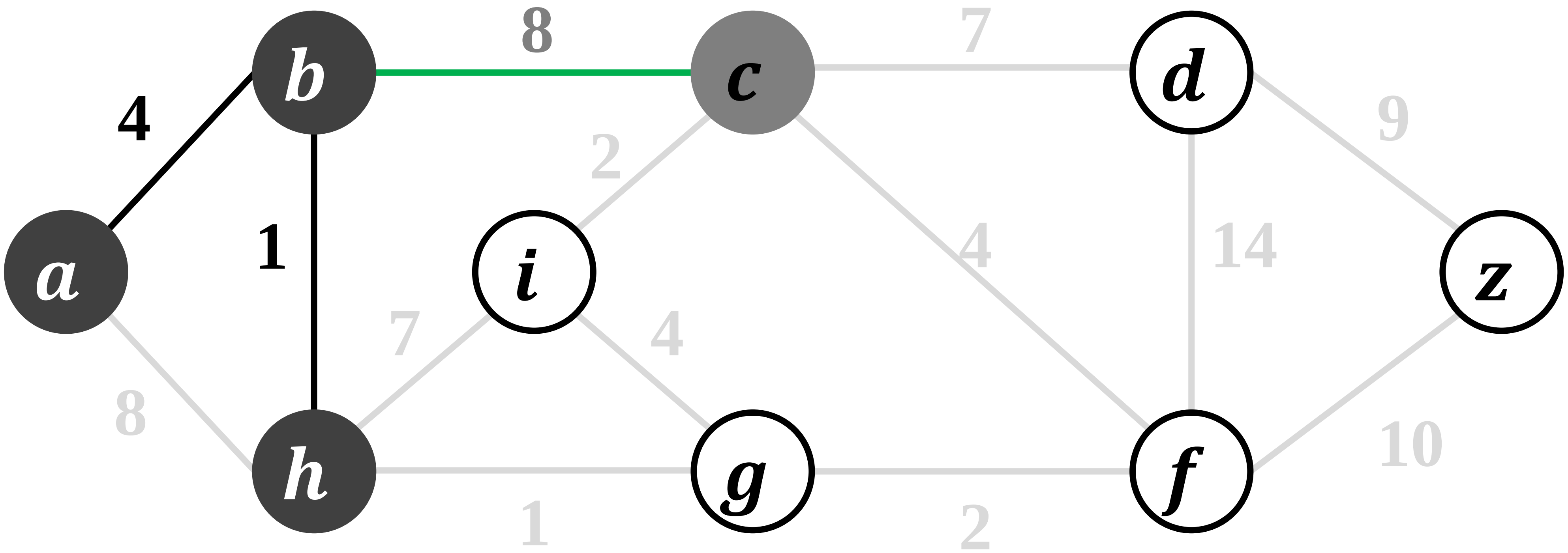
算法实例

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



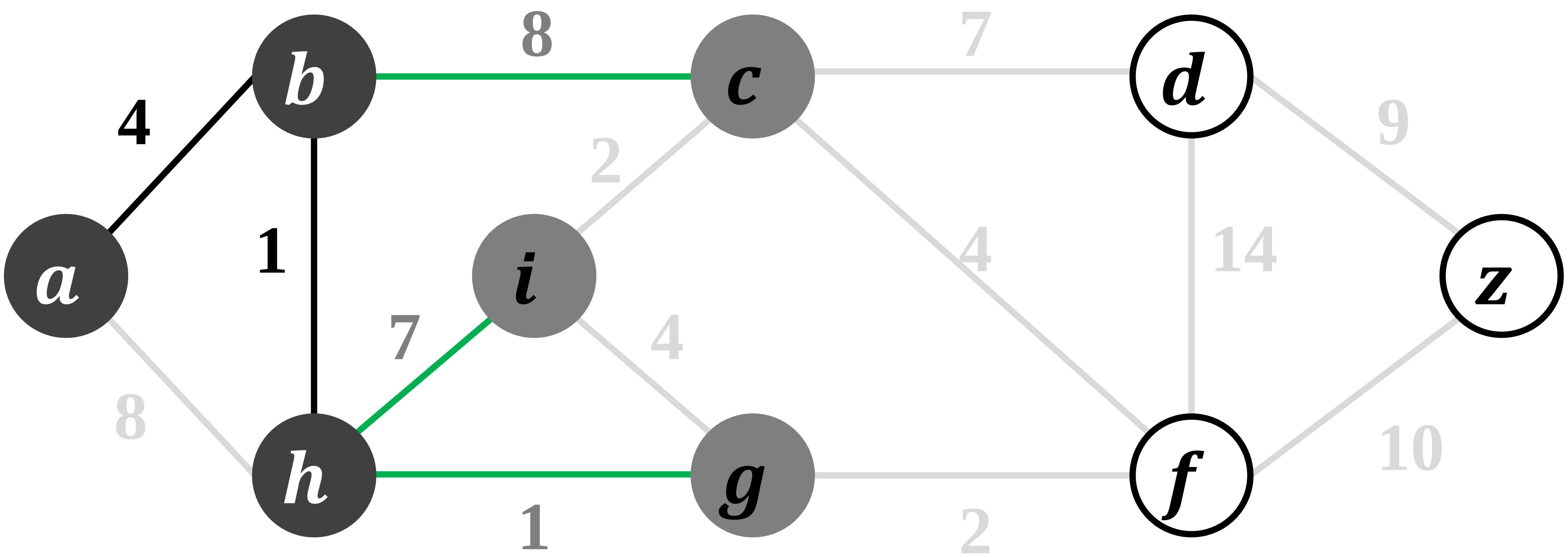
算法实例

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



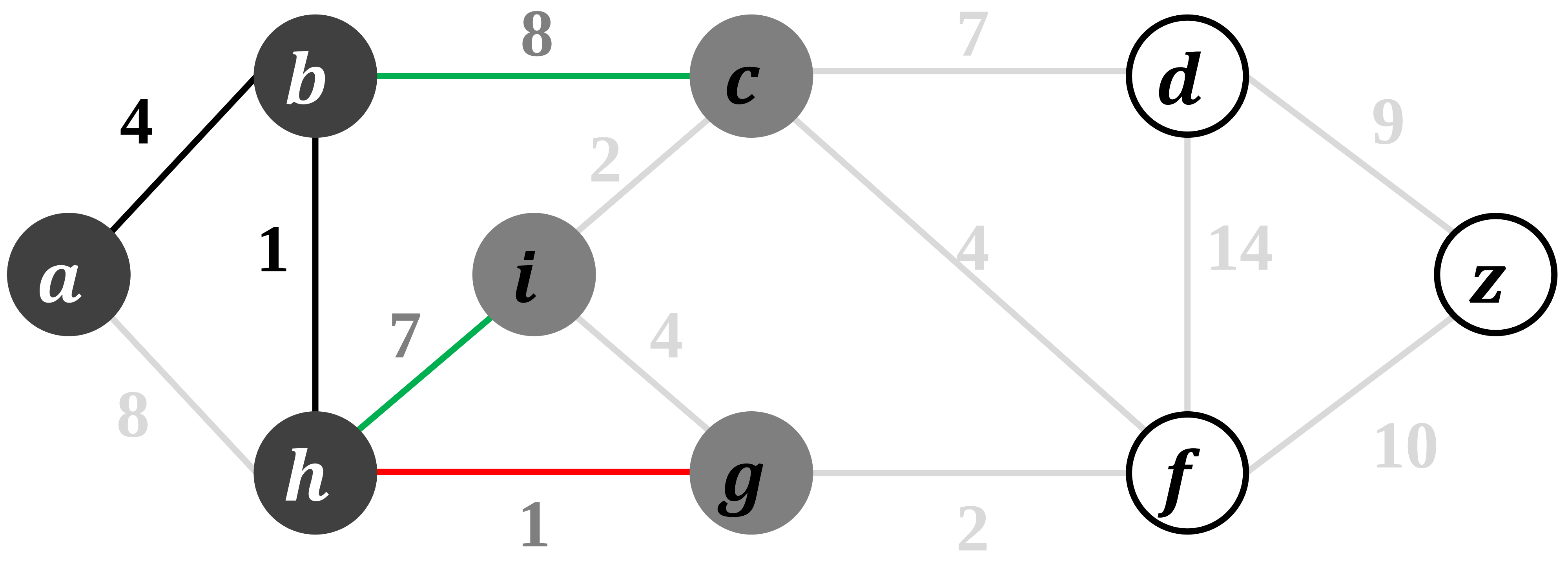
算法实例

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



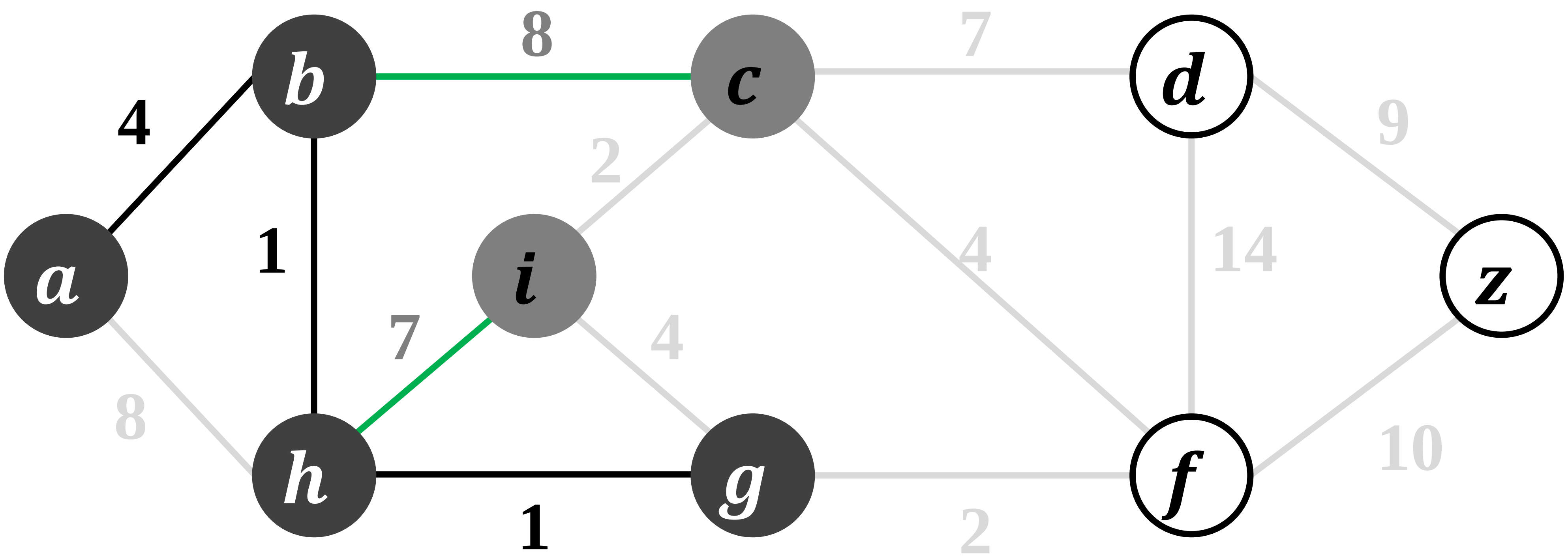
算法实例

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



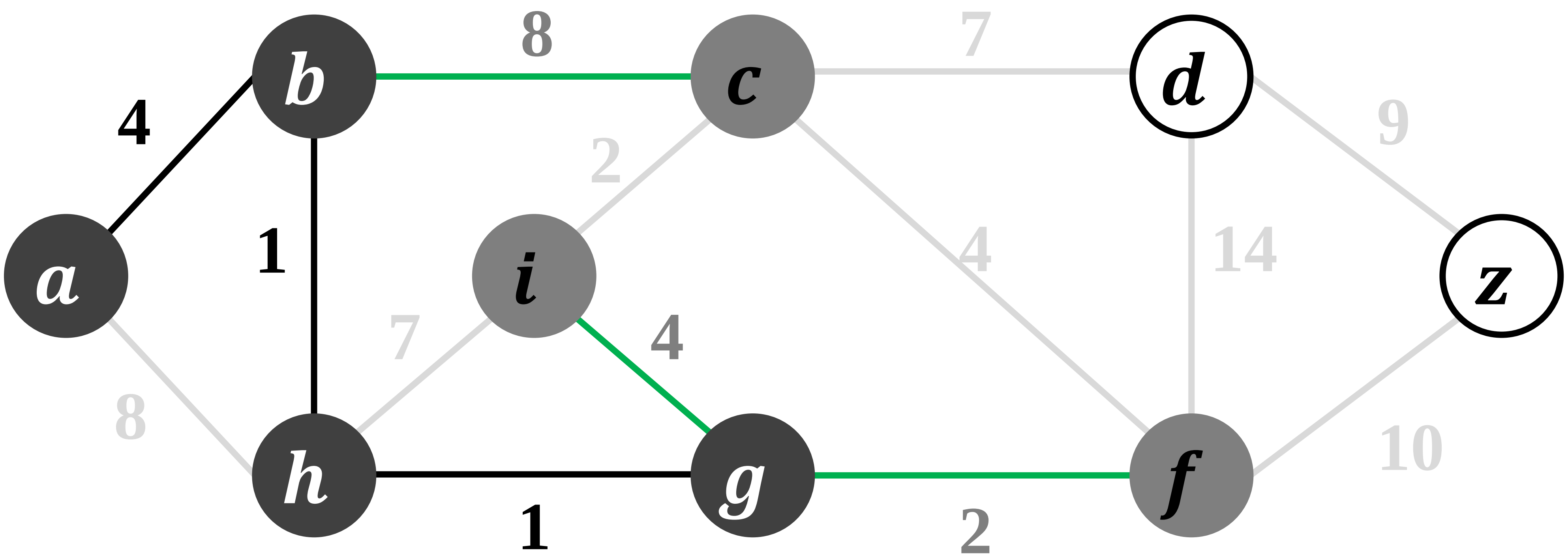
算法实例

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



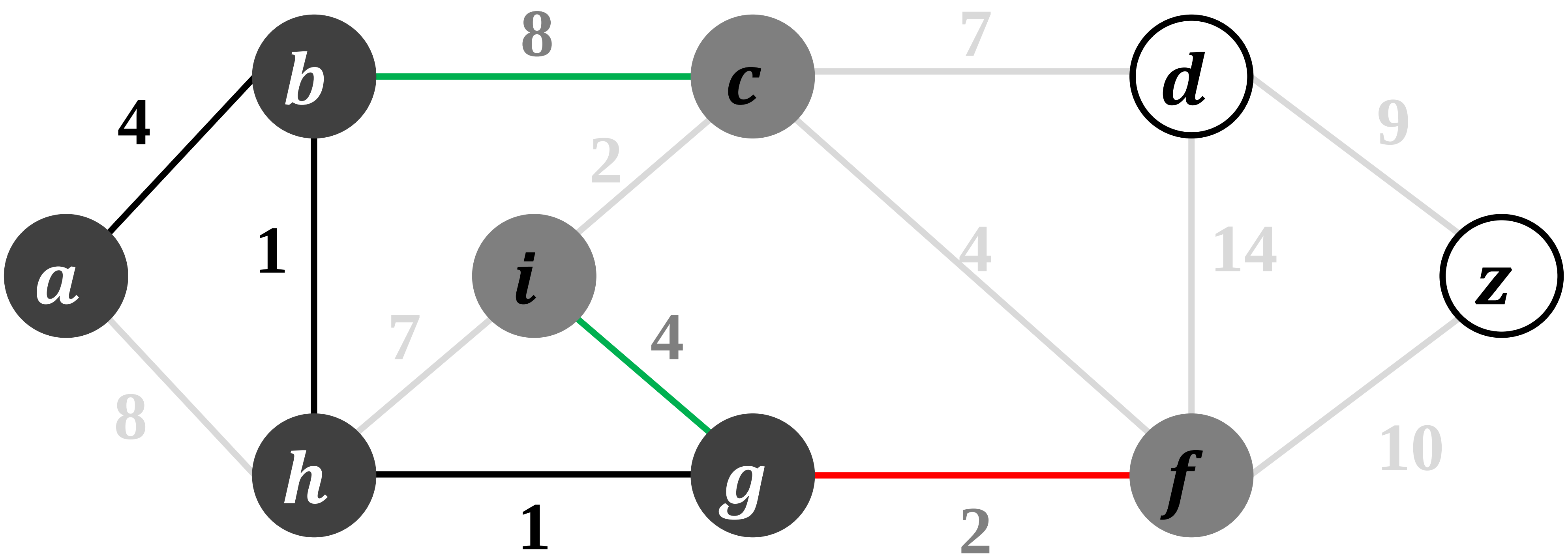
算法实例

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



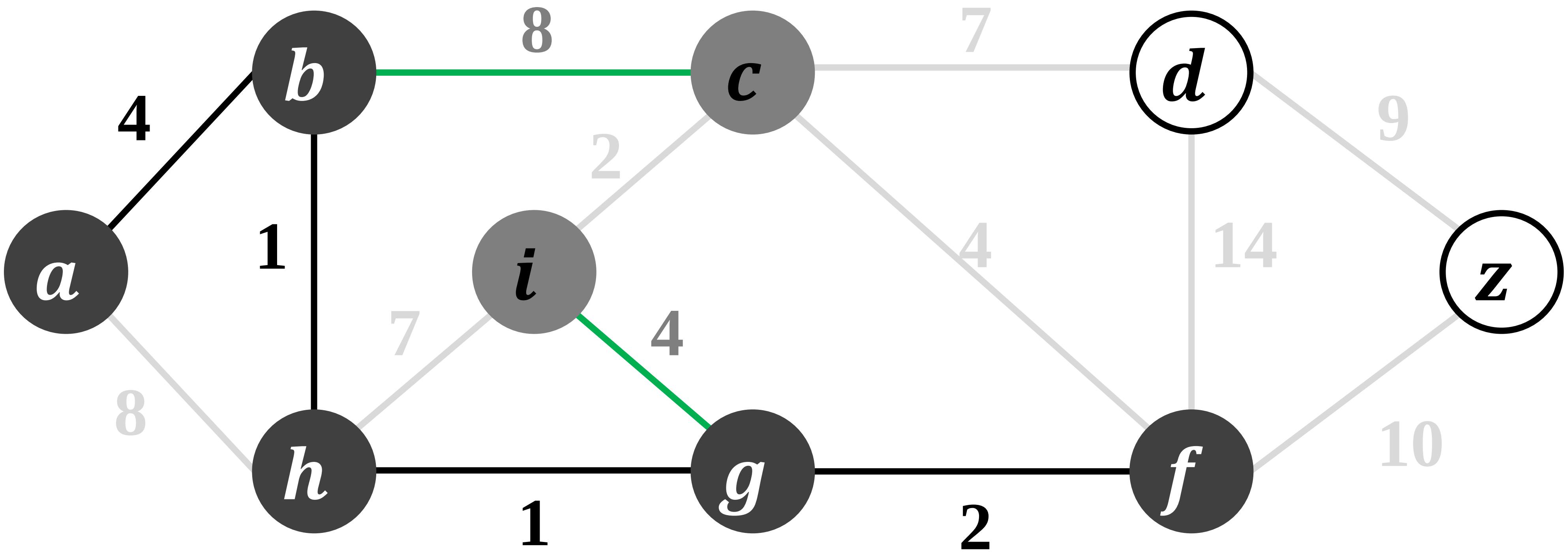
算法实例

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



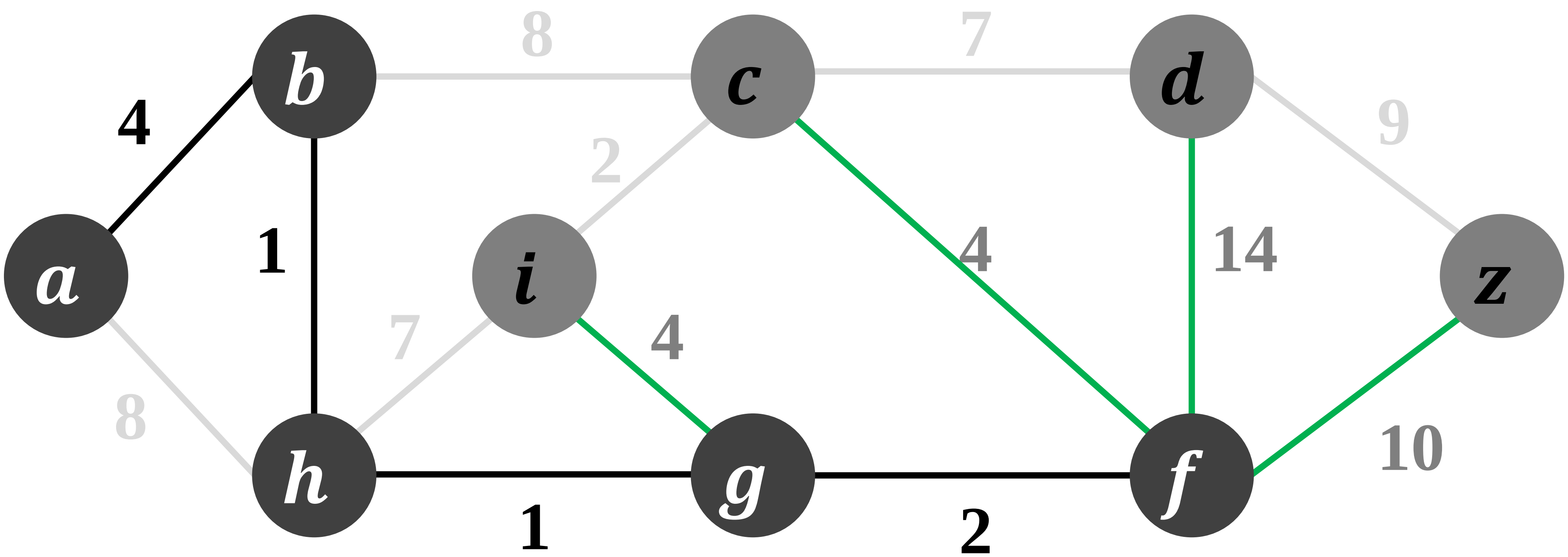
算法实例

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



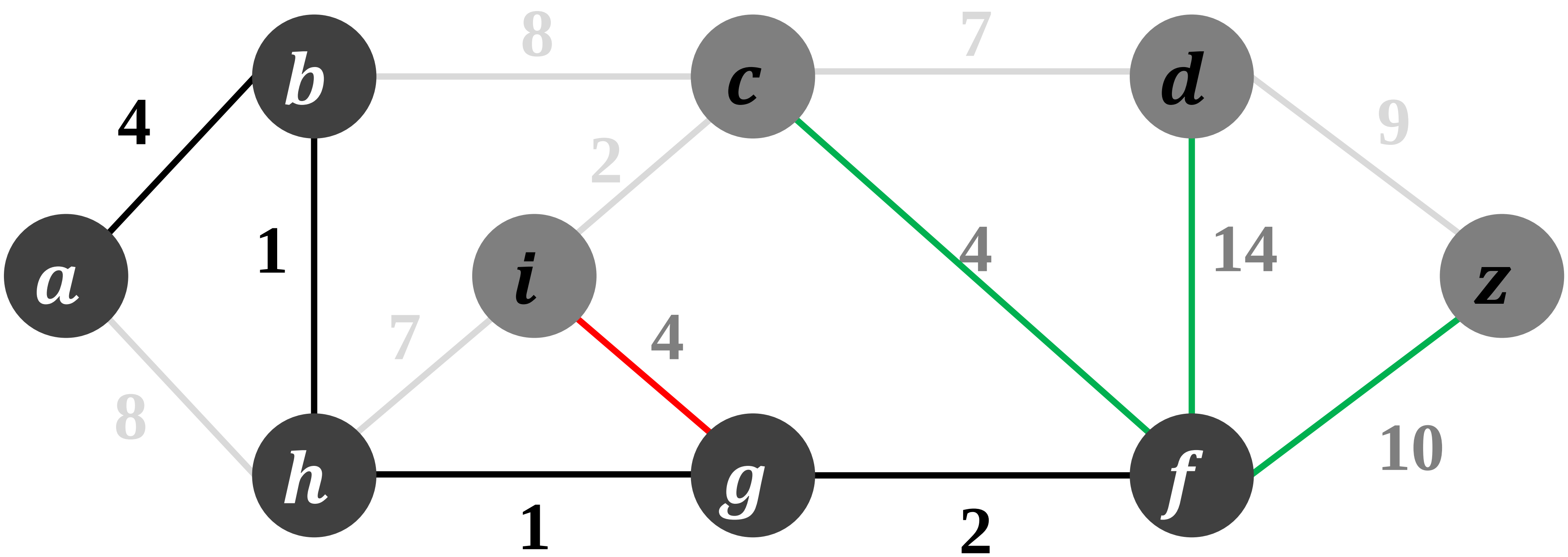
算法实例

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



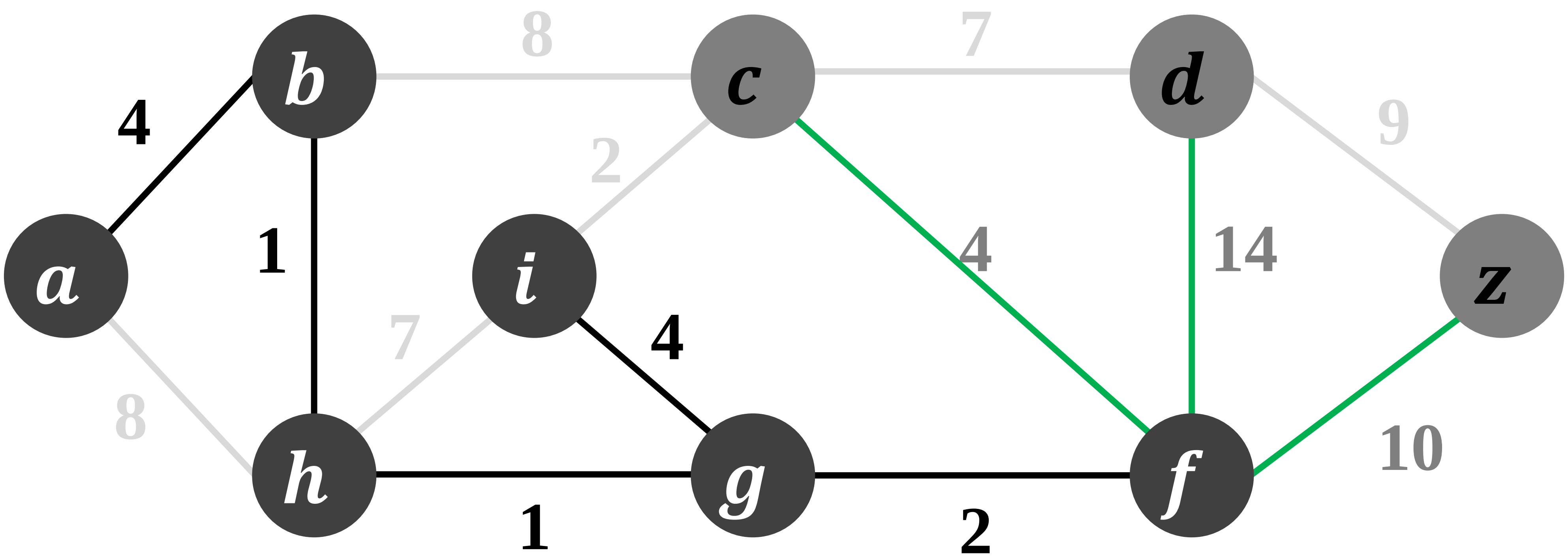
算法实例

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



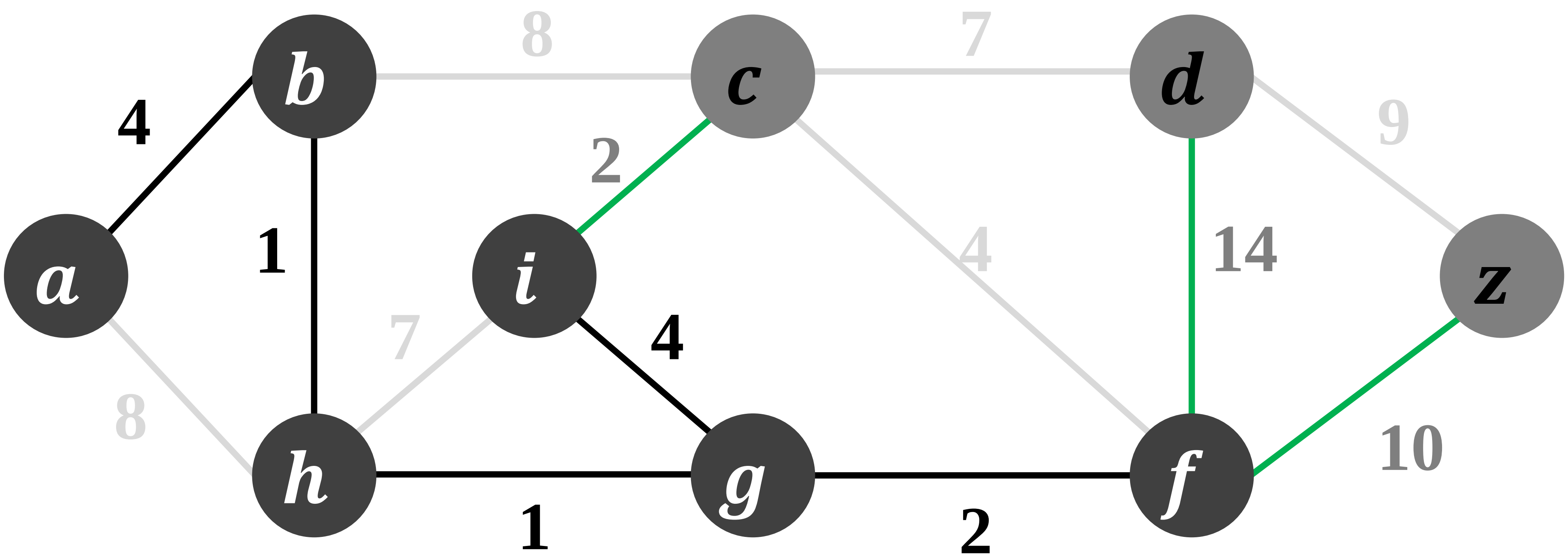
算法实例

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



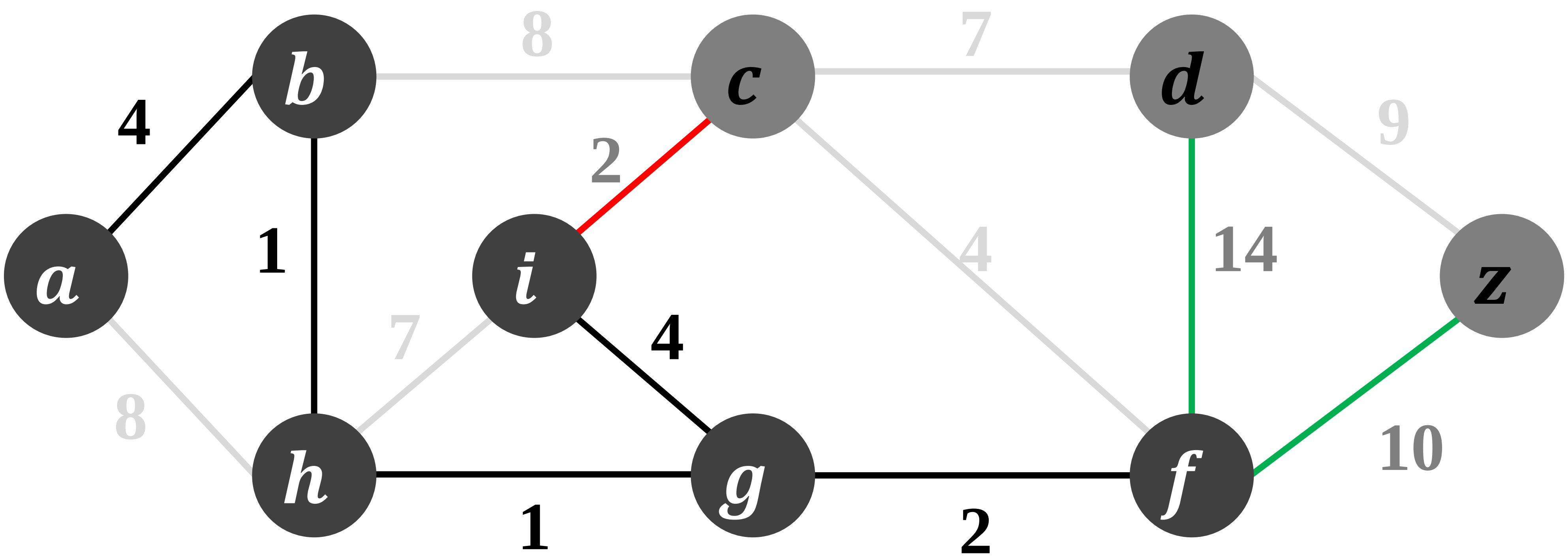
算法实例

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



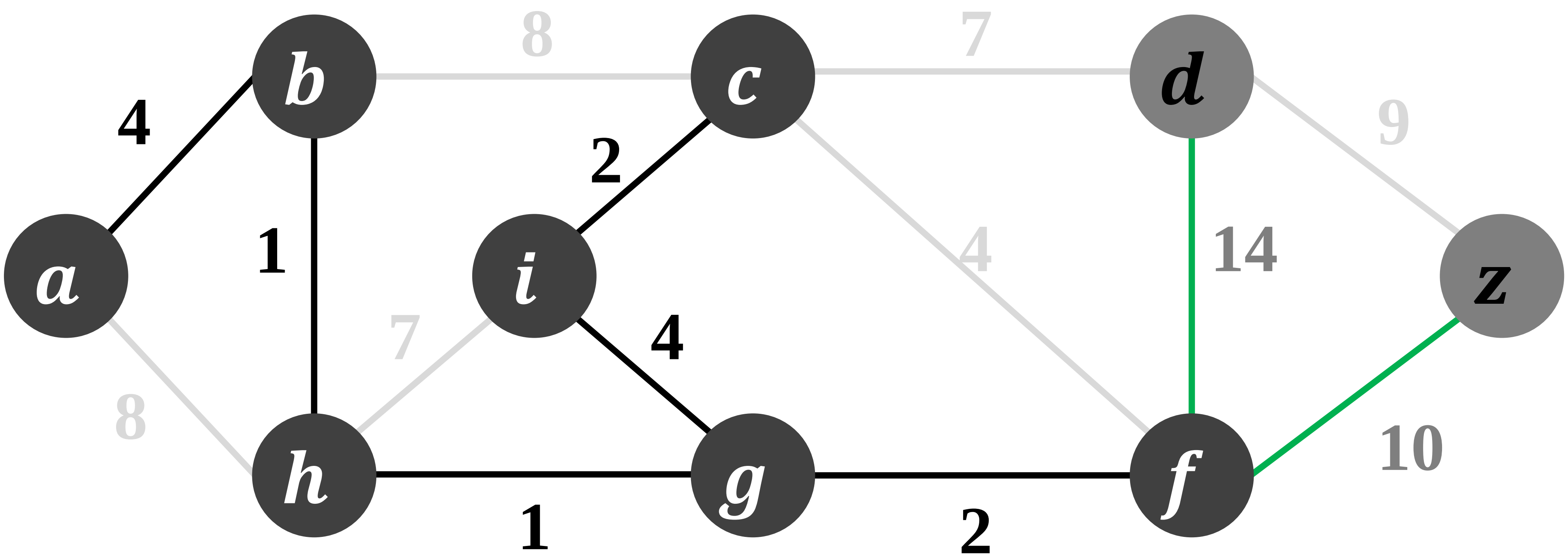
算法实例

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



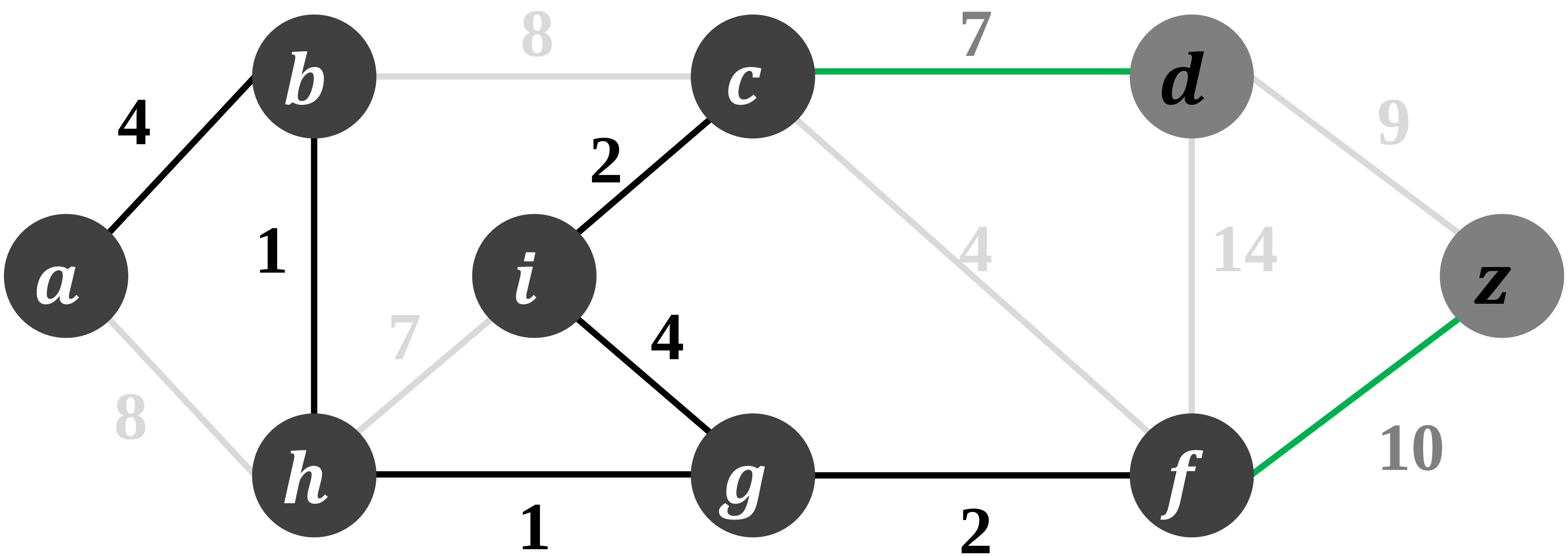
算法实例

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



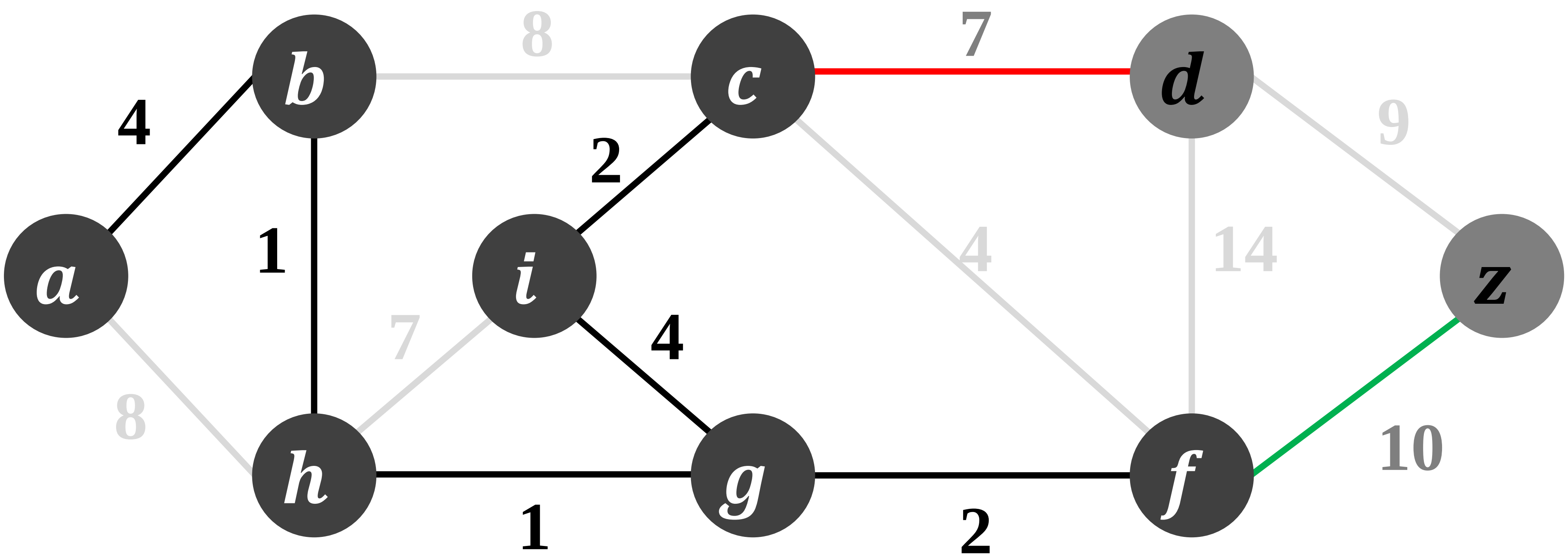
算法实例

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



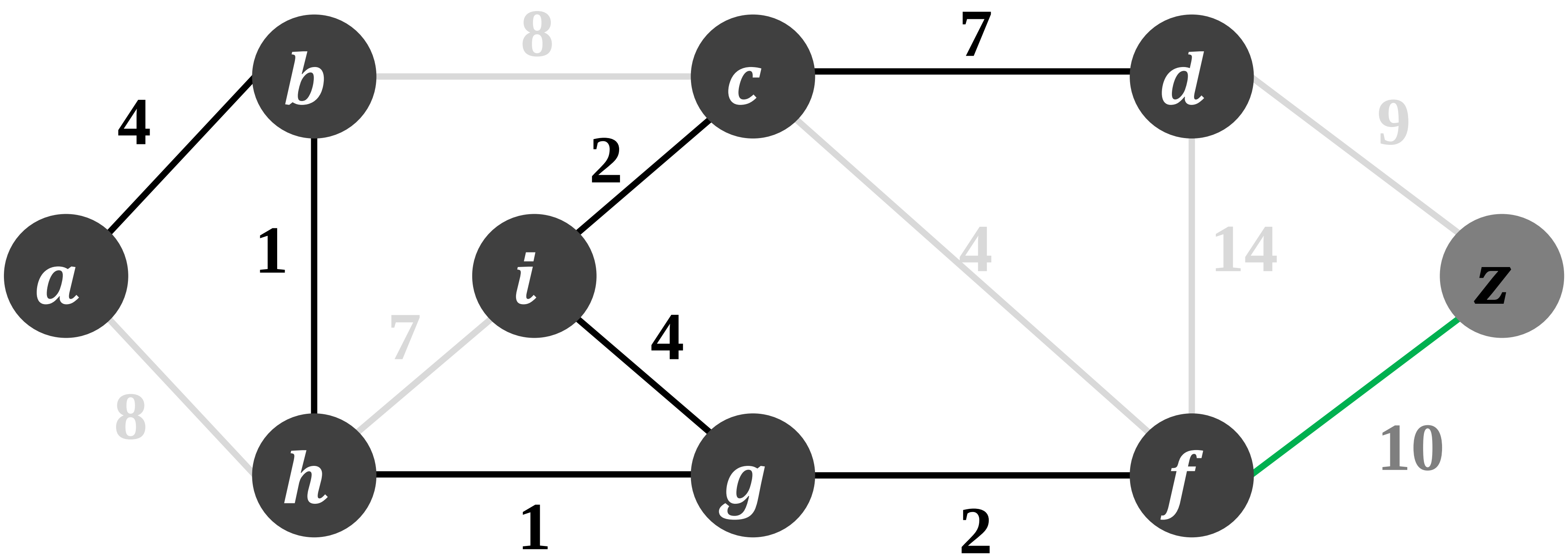
算法实例

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



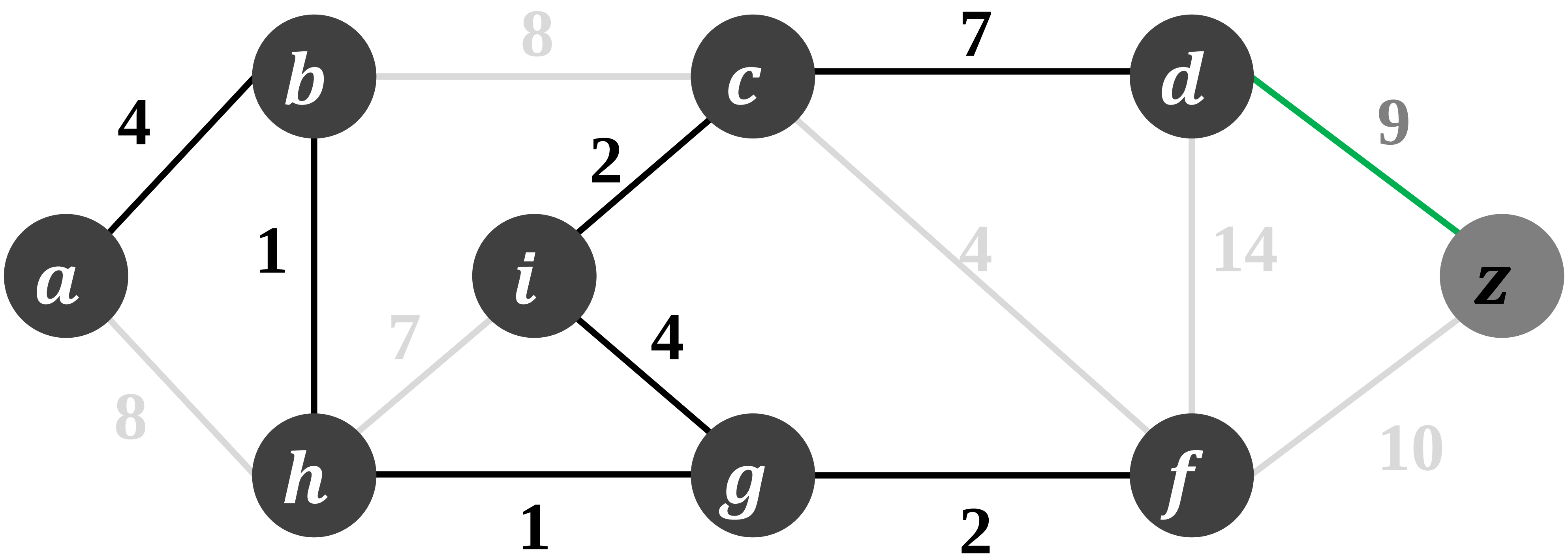
算法实例

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



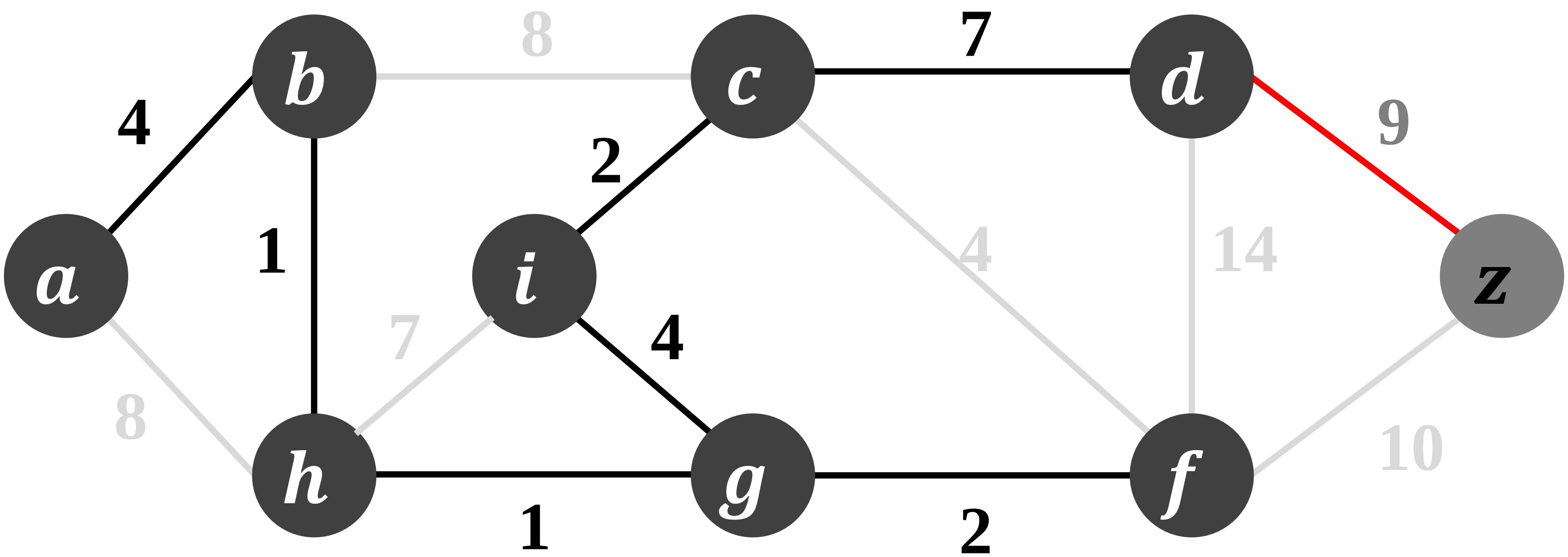
算法实例

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



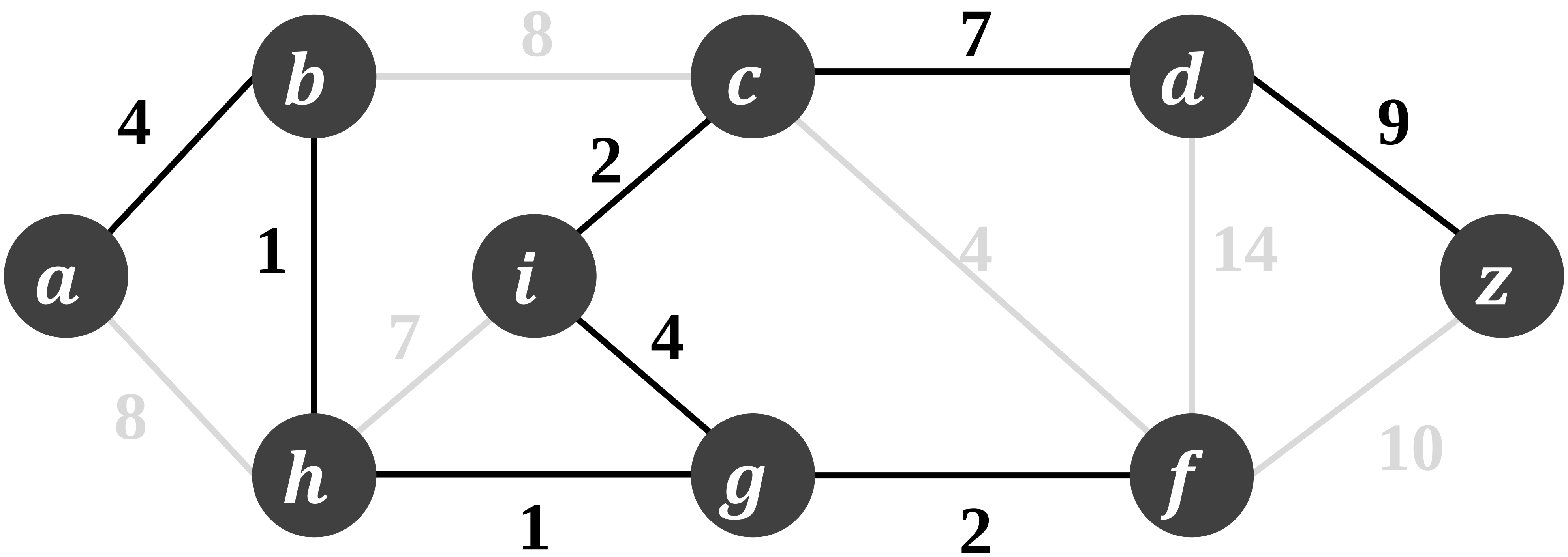
算法实例

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



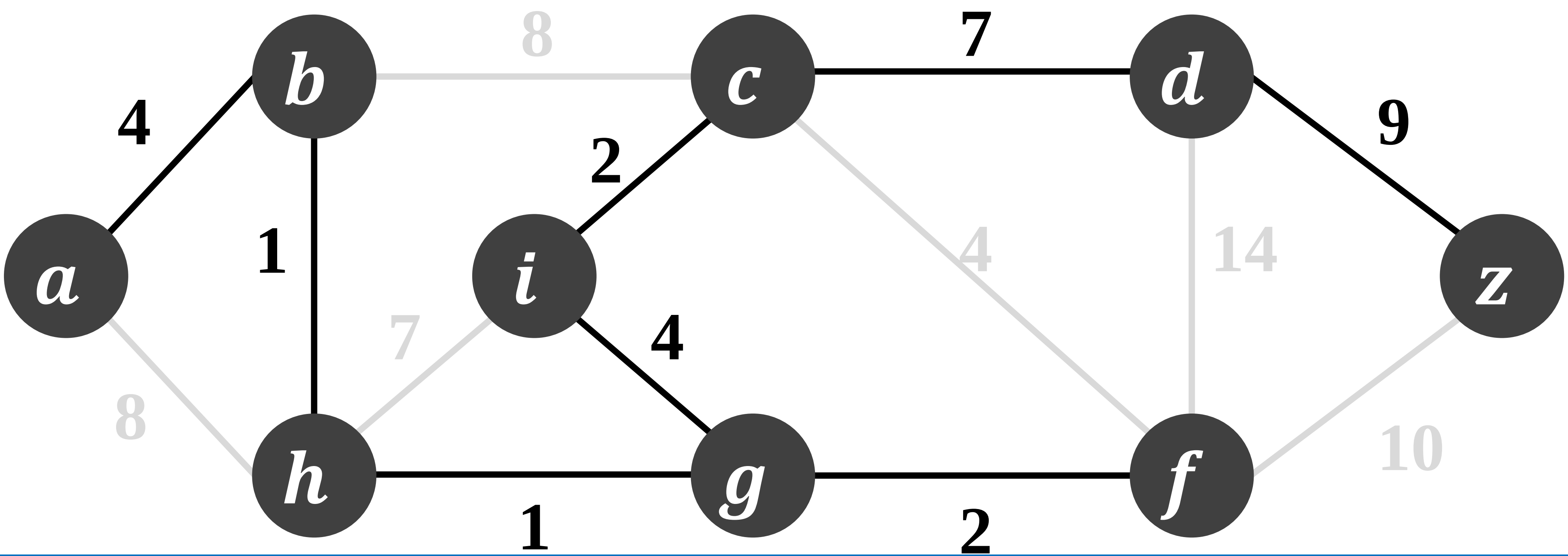
算法实例

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



算法实例

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



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

提纲

- 问题背景
- 通用框架
- Prim算法
- 算法实例
- **算法分析**

伪代码

- **MST-Prim(G)**

输入: 图$G = < V, E, W >$
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

伪代码

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

```
输入: 图G =< V, E, W >
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 ∈ V do
    color[u] ← WHITE
    dist[u] ← ∞
    pred[u] ← NULL
end
dist[1] ← 0
```

选择任意顶点作为起点

伪代码

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

```
//执行最小生成树算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if w(rec, u) < dist[u] then
            dist[u] ← w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

依次添加其他顶点

伪代码

- **MST-Prim(G)**

```
//执行最小生成树算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if w(rec, u) < dist[u] then
            dist[u] ← w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

记录最小权值

伪代码

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

```
//执行最小生成树算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if w(rec, u) < dist[u] then
            dist[u] ← w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

记录安全边的端点

伪代码

- **MST-Prim(G)**

```
//执行最小生成树算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if w(rec, u) < dist[u] then
            dist[u] ← w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

记录新增的安全边

伪代码

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

```
//执行最小生成树算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if w(rec, u) < dist[u] then
            dist[u] ← w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

更新$dist$数组

伪代码

- **MST-Prim(G)**

```
//执行最小生成树算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if w(rec, u) < dist[u] then
            dist[u] ← w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

标记顶点处理完成

复杂度分析

- **MST-Prim(G)**

输入: 图$G=<V,E,W>$
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

$$\left.\begin{array}{l}
\textbf{for } u \in V \textbf{ do} \\
\quad color[u] \leftarrow WHITE \\
\quad dist[u] \leftarrow \infty \\
\quad pred[u] \leftarrow NULL \\
\textbf{end} \\
dist[1] \leftarrow 0
\end{array}\right\} O(|V|)$$

复杂度分析

- **MST-Prim(G)**

```
//执行最小生成树算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if w(rec, u) < dist[u] then
            dist[u] ← w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

$O(|V|)$

$O(\deg(u))$

复杂度分析

- **MST-Prim(G)**

```
//执行最小生成树算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if w(rec, u) < dist[u] then
            dist[u] ← w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

$O(|V|)$

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

$O(\deg(u))$

复杂度分析

- **MST-Prim(G)**

```
//执行最小生成树算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if w(rec, u) < dist[u] then
            dist[u] ← w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

$\boldsymbol{O}(|V|)$

$\boldsymbol{O}(|V| \cdot |V|) = \boldsymbol{O}(|V|^2)$

$\boldsymbol{O}(\deg(u))$

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

复杂度分析

- **MST-Prim(G)**

```
//执行最小生成树算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if w(rec, u) < dist[u] then
            dist[u] ← w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

$O(|V|)$

$O(\deg(u))$

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

$O(|V|^2)$

复杂度分析

- **MST-Prim(G)**

```
//执行最小生成树算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if w(rec, u) < dist[u] then
            dist[u] ← w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

$O(|V|)$

$O(\deg(u))$

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

$O(|V|^2)$

问题：能否进一步优化？

复杂度分析

- **MST-Prim(G)**

//执行最小生成树算法

```
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if w(rec, u) < dist[u] then
            dist[u] ← w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

$O(|V|)$ → $O(?)$

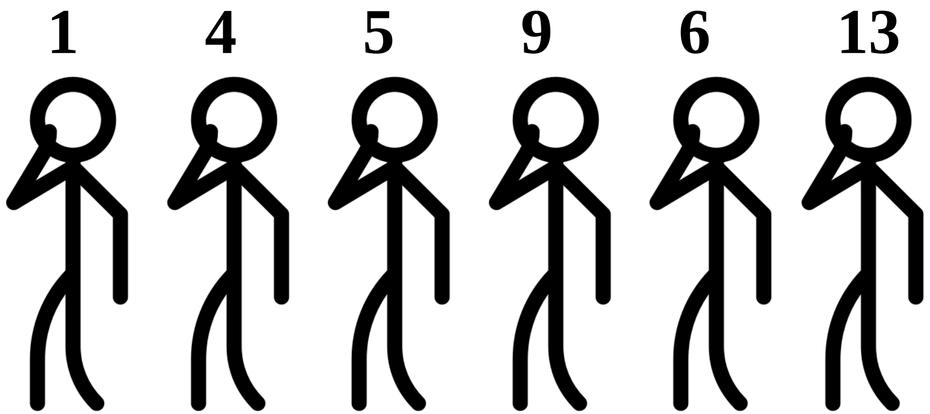
$O(|V|^2)$

数据结构加速最小值查询

数据结构：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

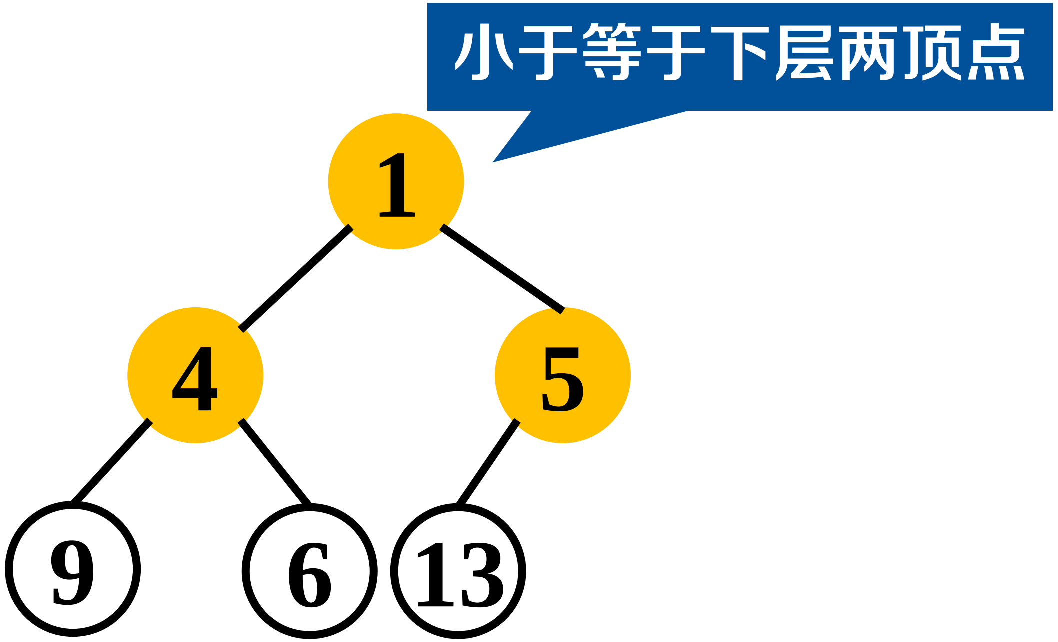


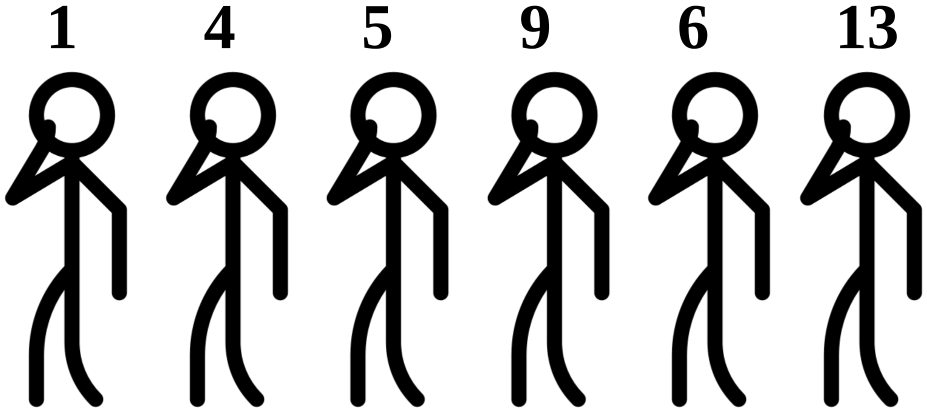
优先队列Q

数据结构：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



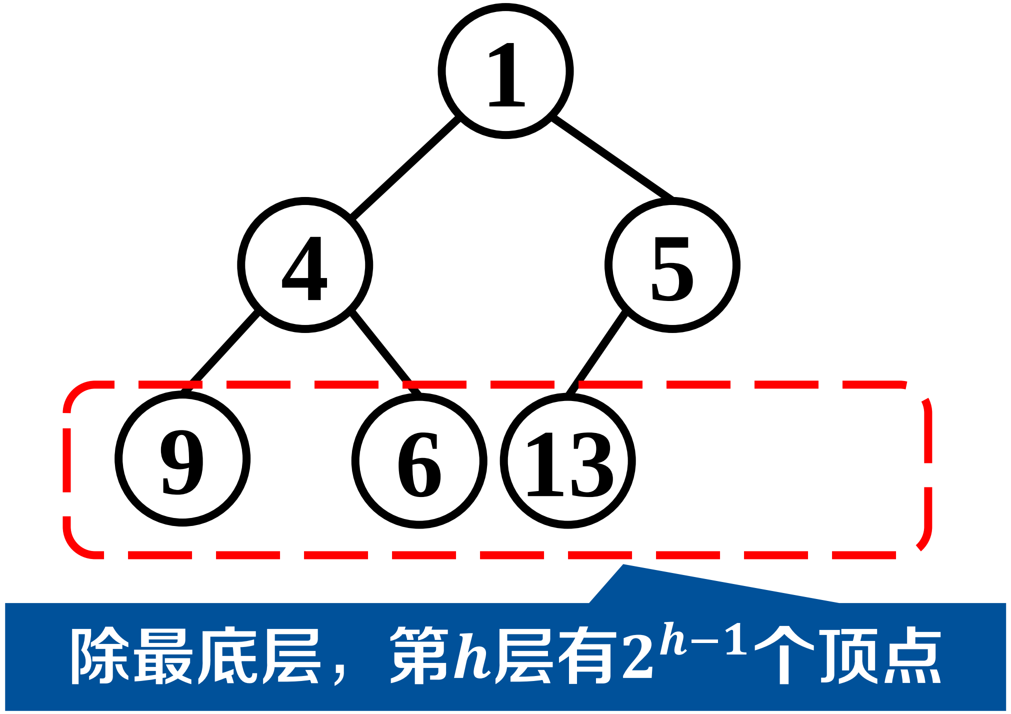


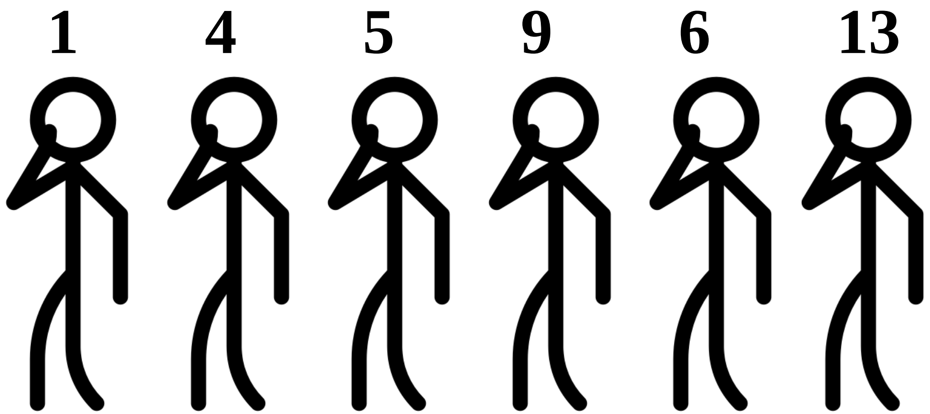
优先队列Q

数据结构：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



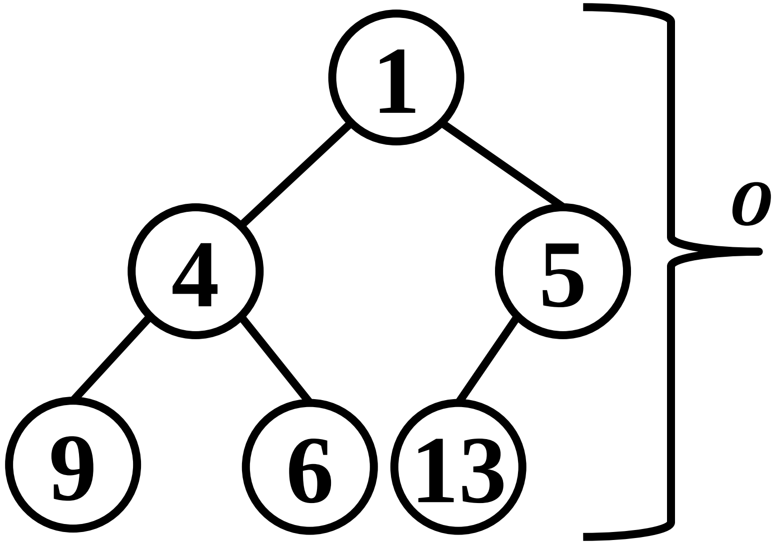


优先队列Q

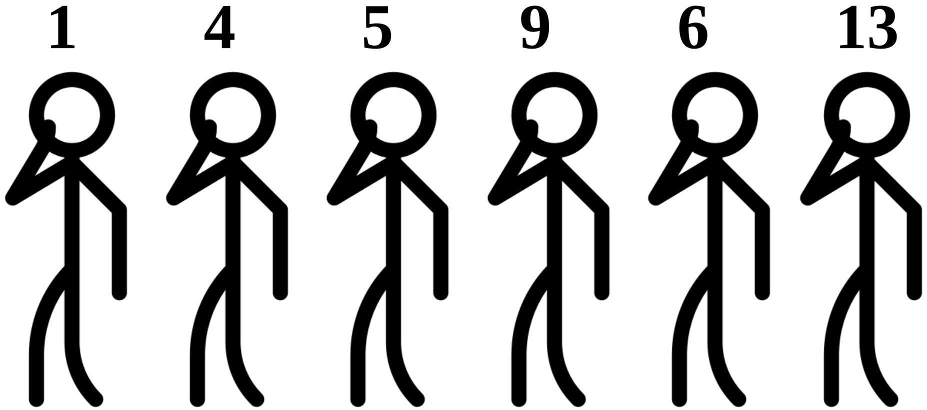
数据结构：优先队列

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



$O(\log n)$层



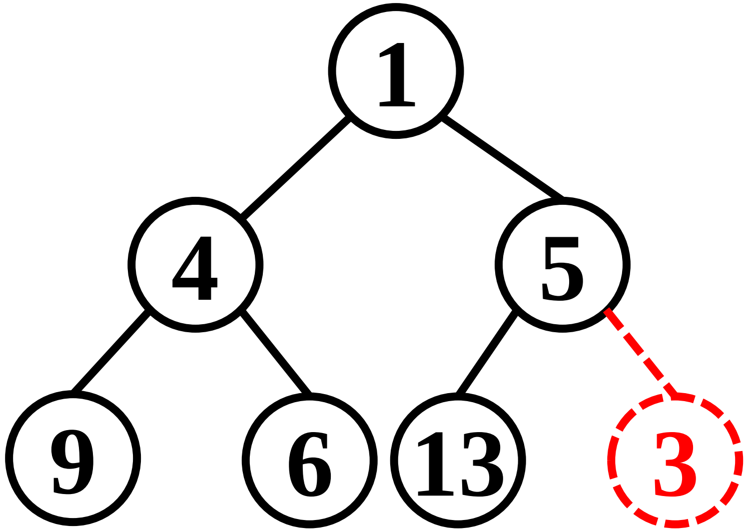
优先队列Q

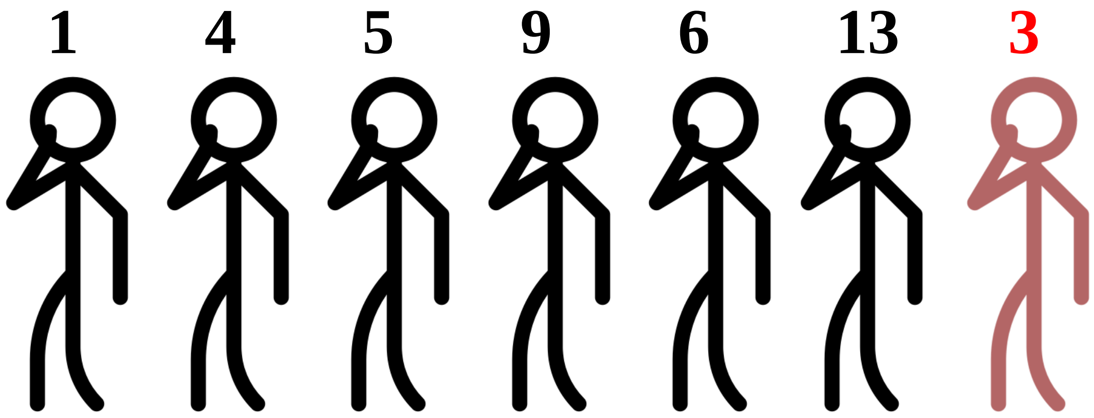
除最底层，第h层有2^{h-1}个顶点

数据结构：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
  - $Q.Insert()$



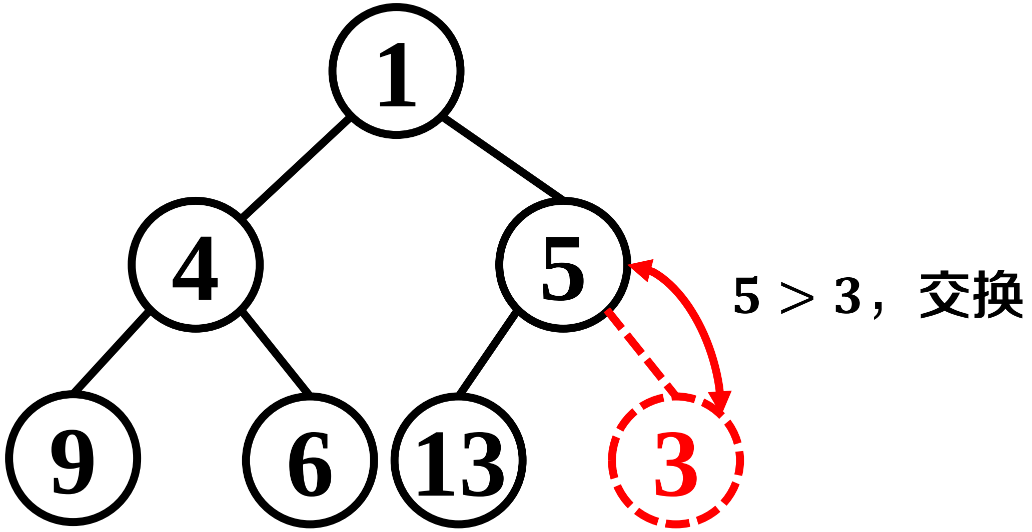


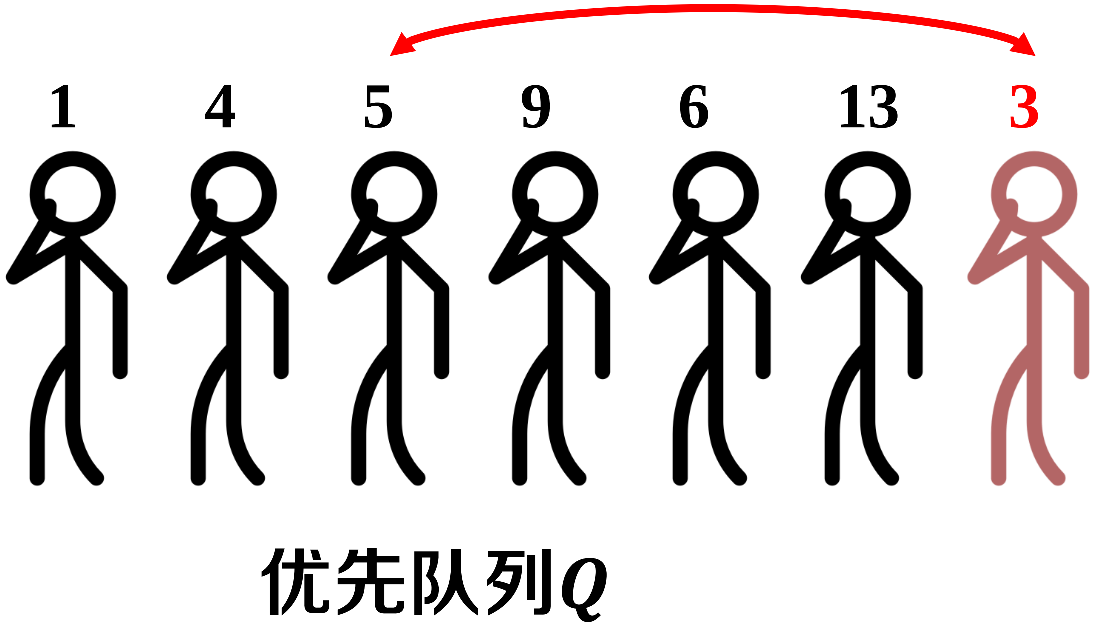
优先队列Q

数据结构：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
  - $Q.Insert()$

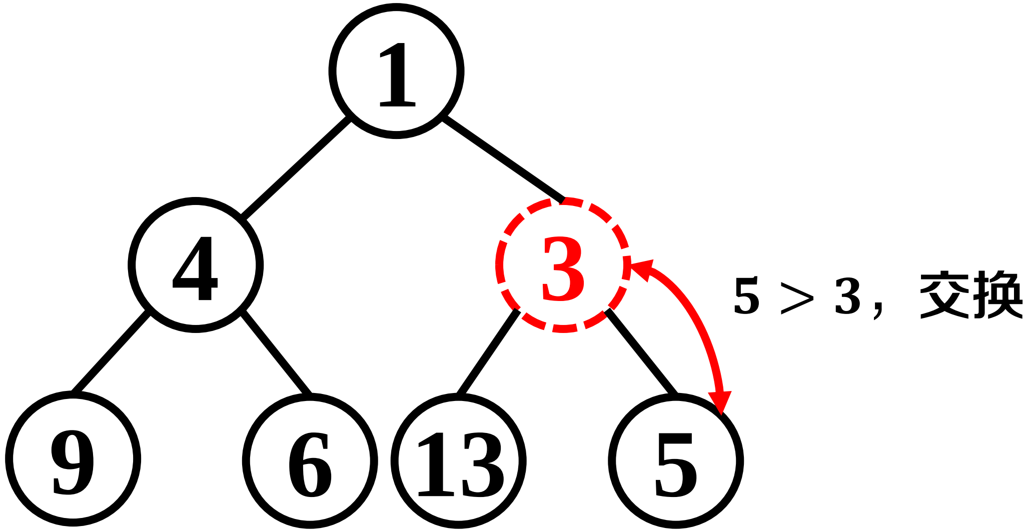


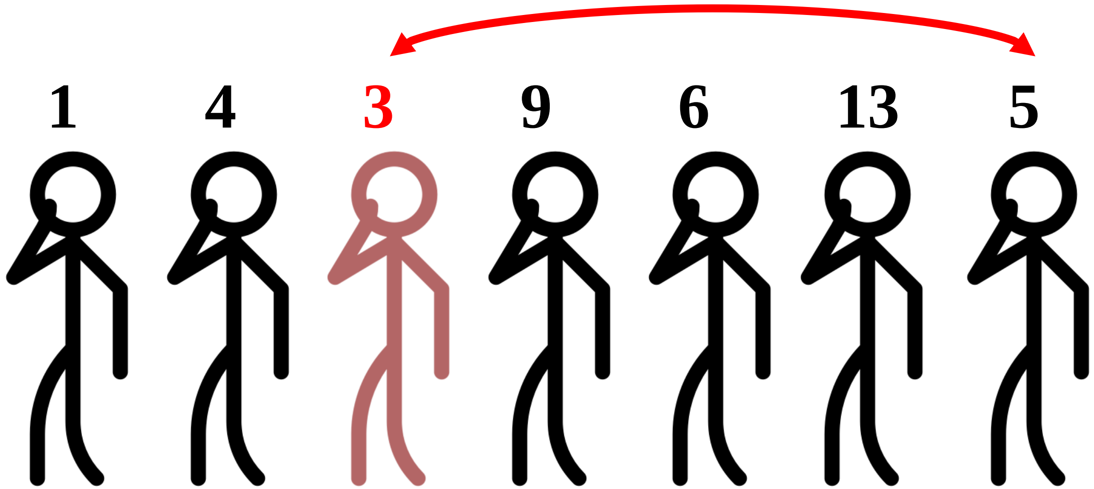


数据结构：优先队列

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
  - $Q.Insert()$



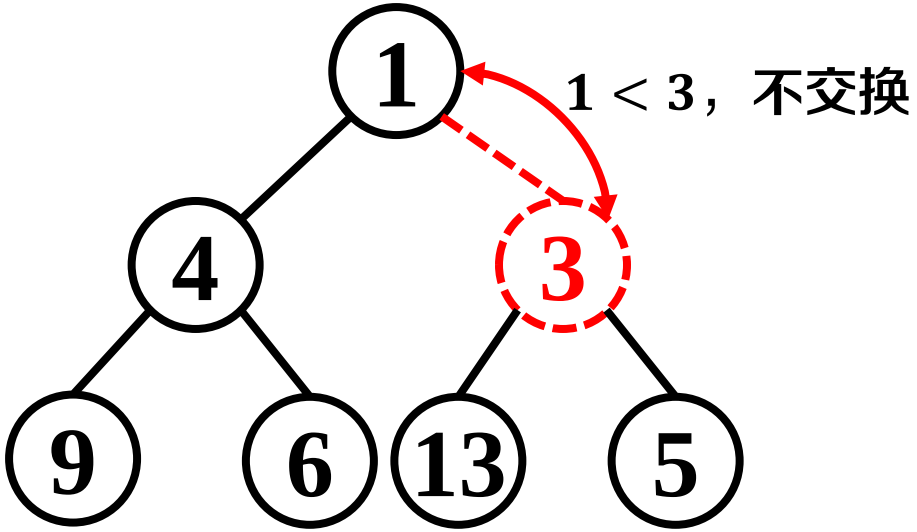


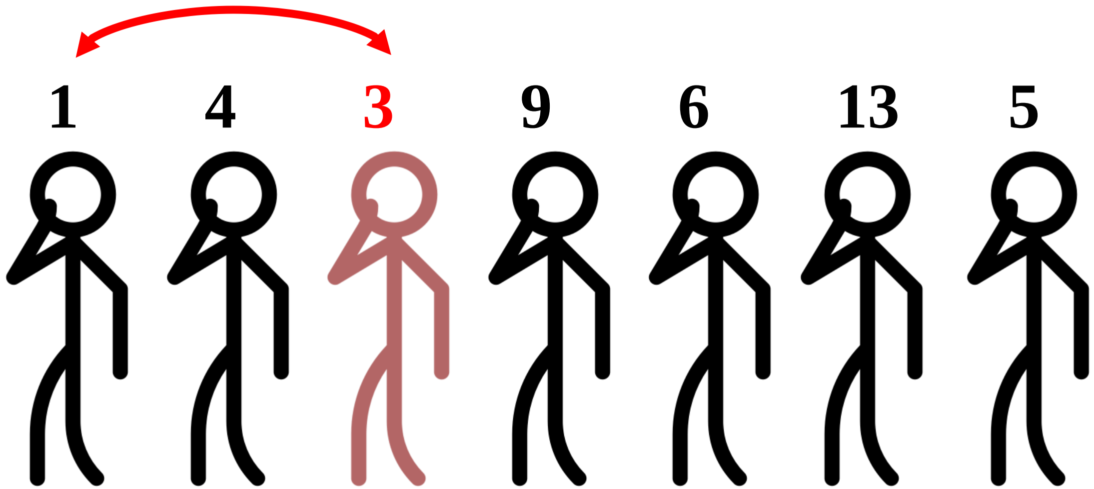
优先队列Q

数据结构：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
  - $Q.Insert()$

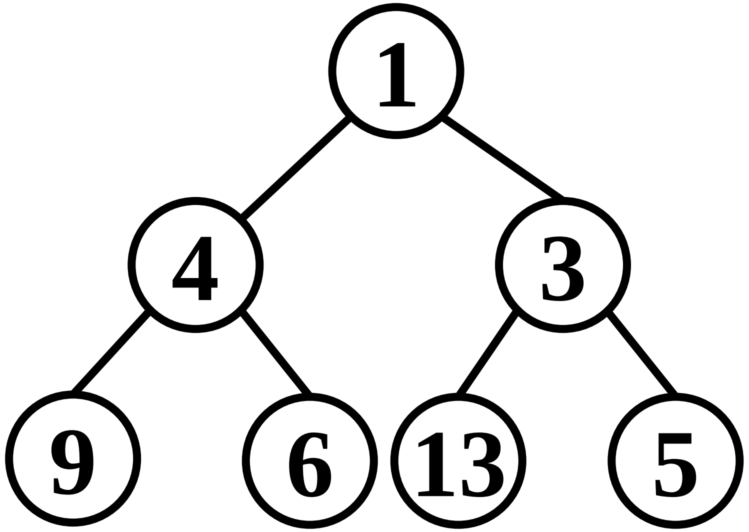


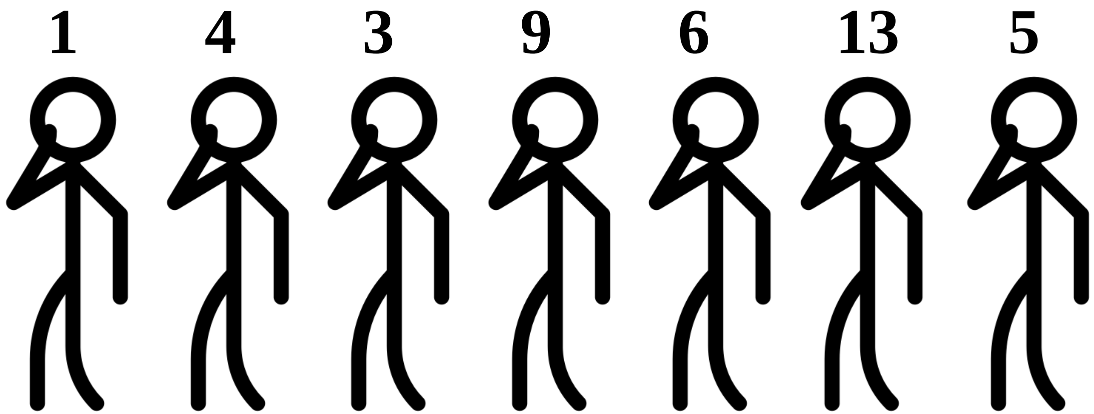


数据结构：优先队列

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
  - $Q.Insert()$



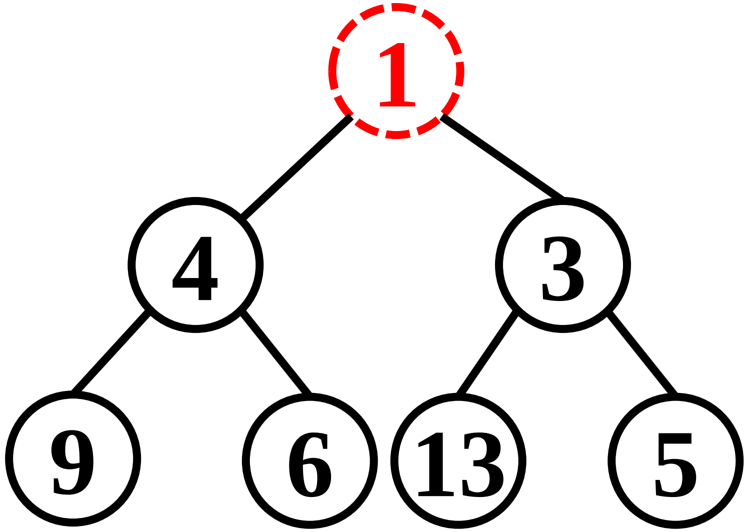


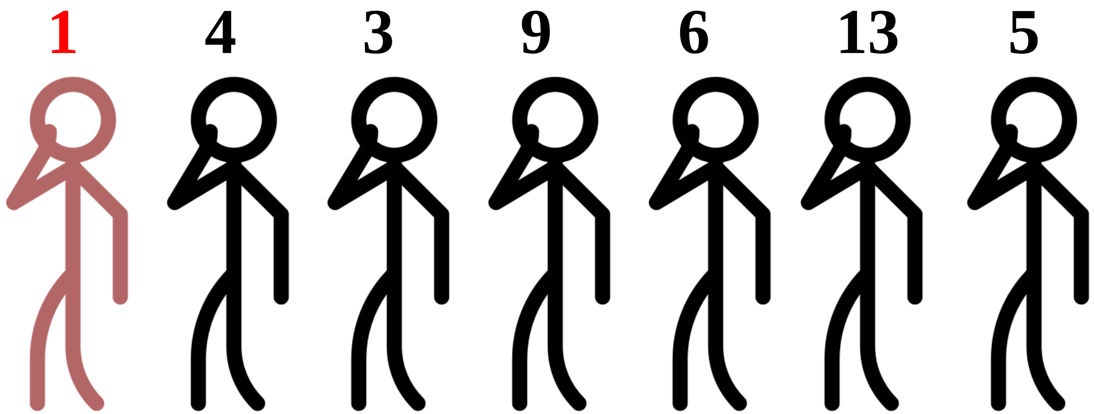
优先队列Q

数据结构：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
  - $Q.Insert()$
  - $Q.ExtractMin()$



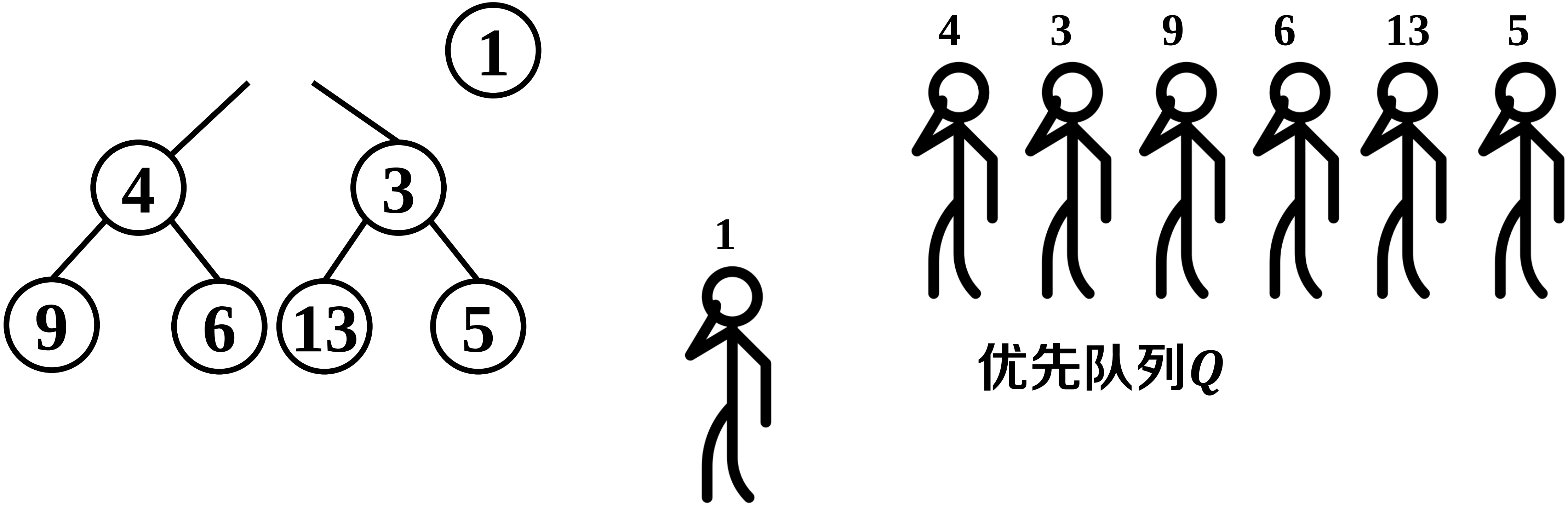


优先队列Q

数据结构：优先队列

优先队列

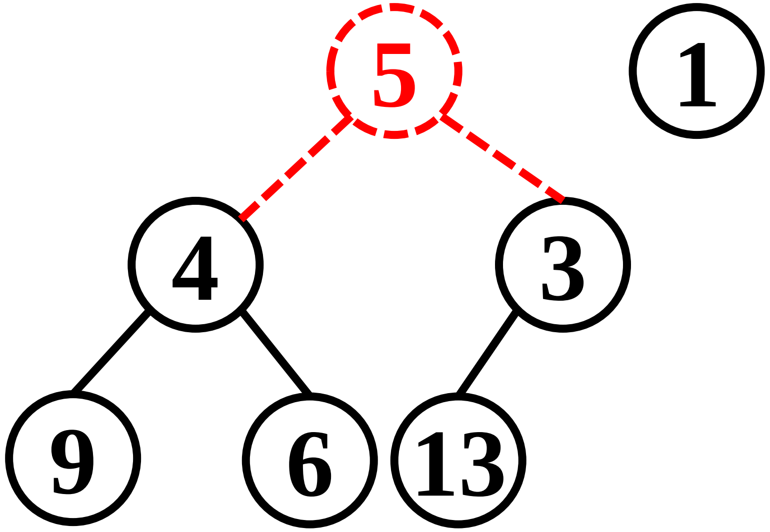
- 队列中每个元素有一个关键字，依据关键字大小离开队列
- 通过二叉堆来实现优先队列
  - $Q.Insert()$
  - $Q.ExtractMin()$

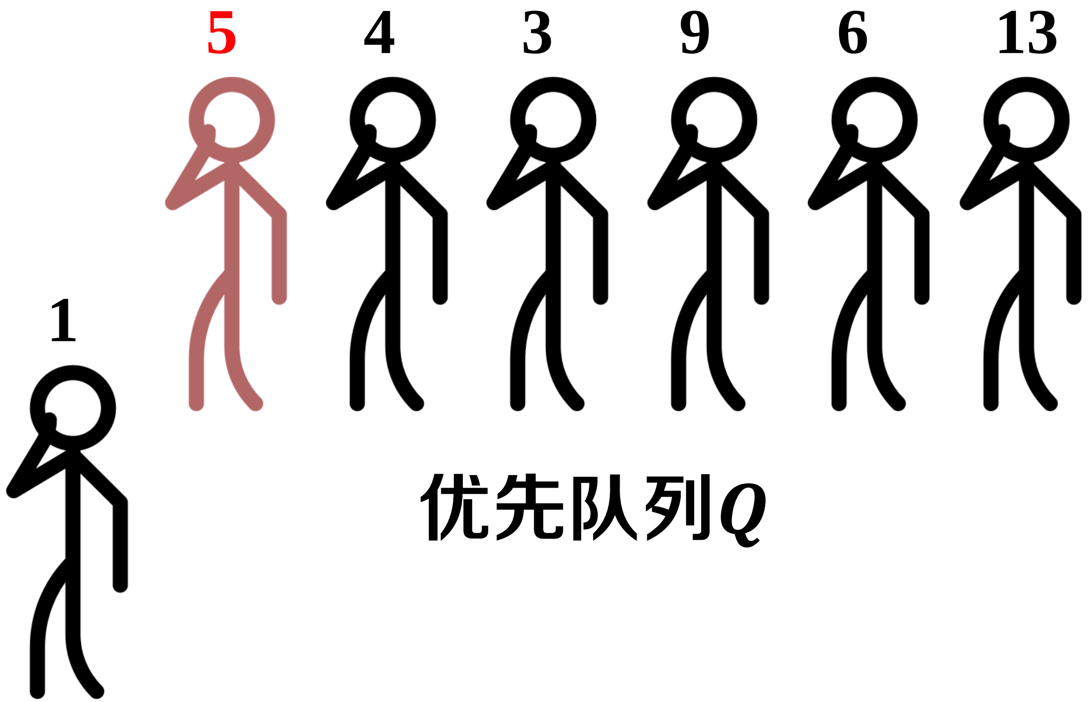


数据结构：优先队列

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
  - $Q.Insert()$
  - $Q.ExtractMin()$

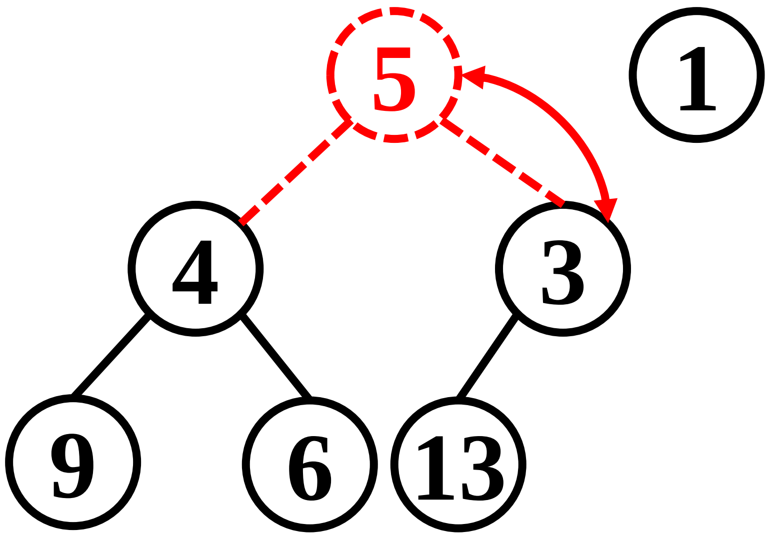


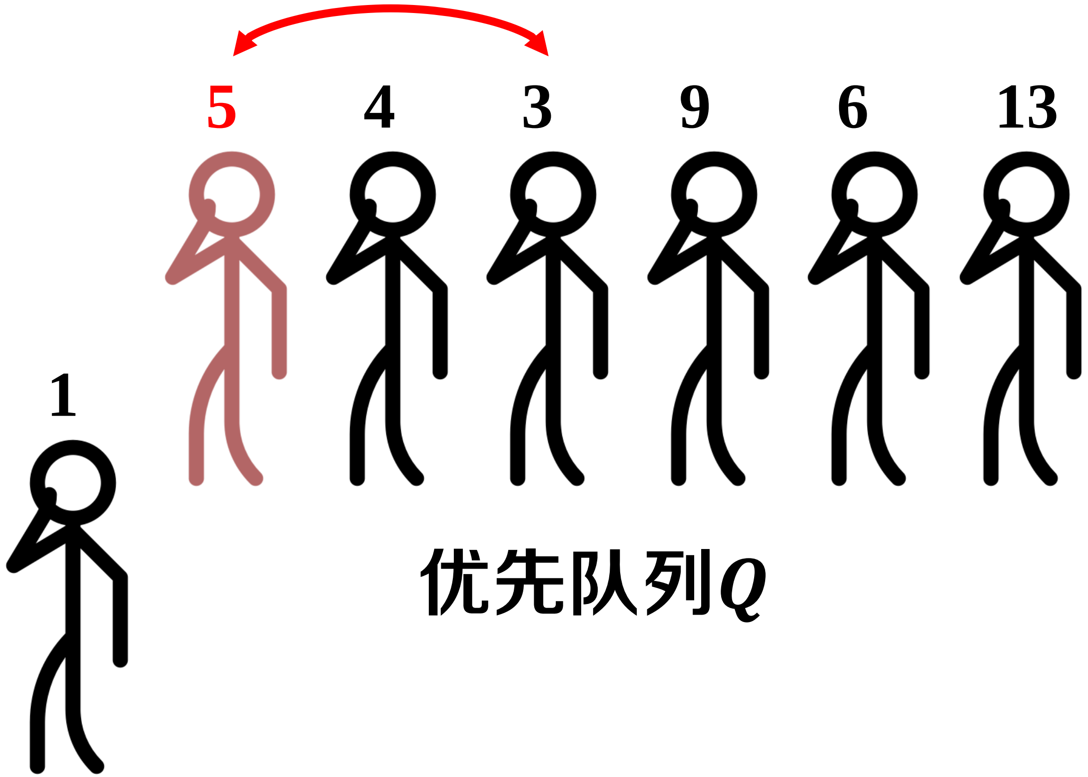


数据结构：优先队列

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
  - $Q.Insert()$
  - $Q.ExtractMin()$



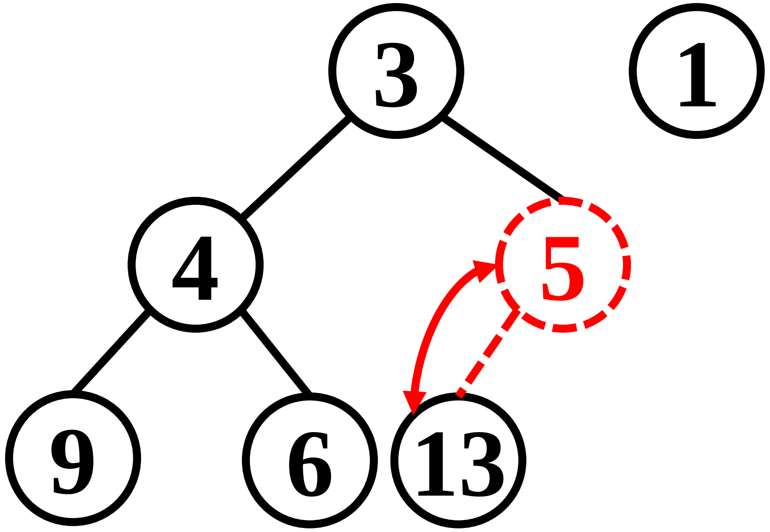


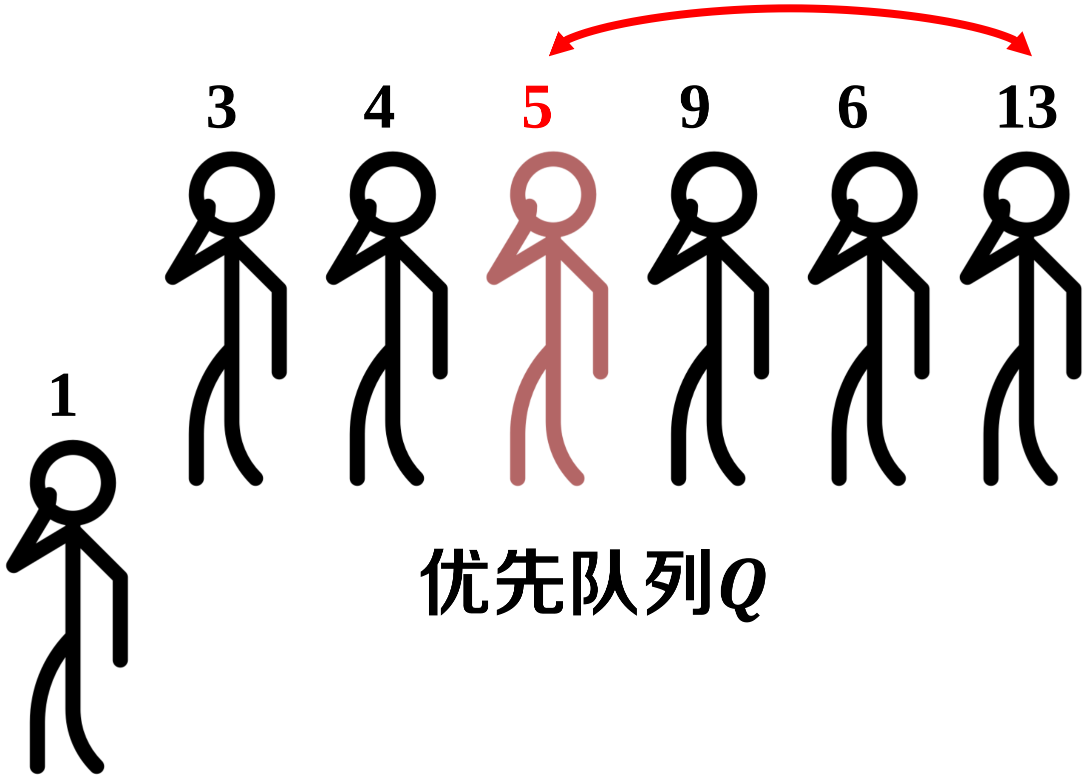
优先队列Q

数据结构：优先队列

优先队列

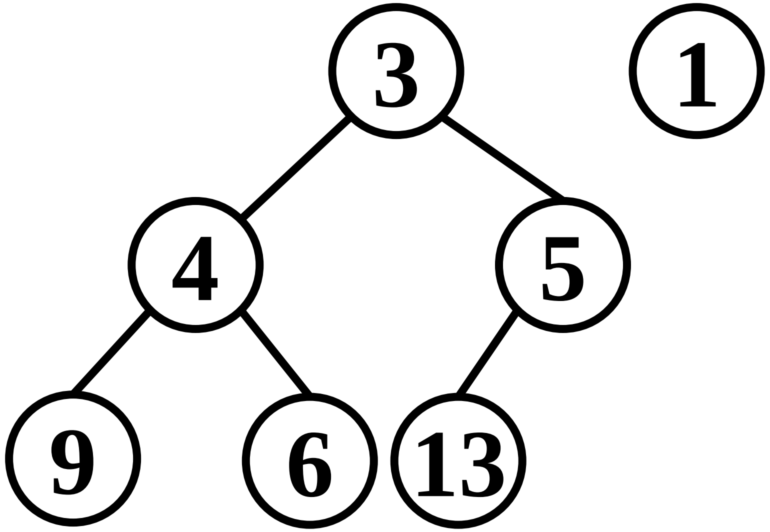
- 队列中每个元素有一个关键字，依据关键字大小离开队列
- 通过二叉堆来实现优先队列
  - $Q.Insert()$
  - $Q.ExtractMin()$

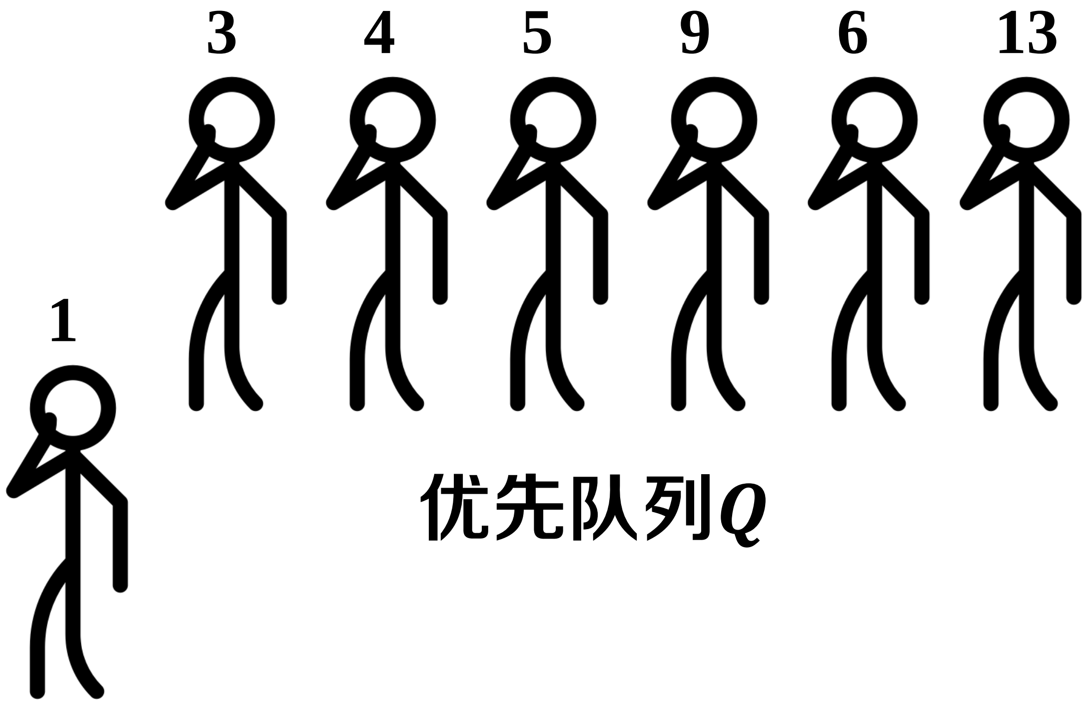




优先队列

- 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
    - $Q.Insert()$
    - $Q.ExtractMin()$

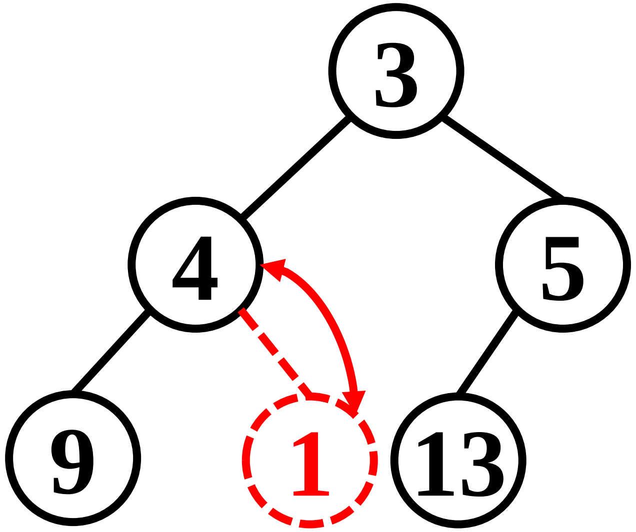


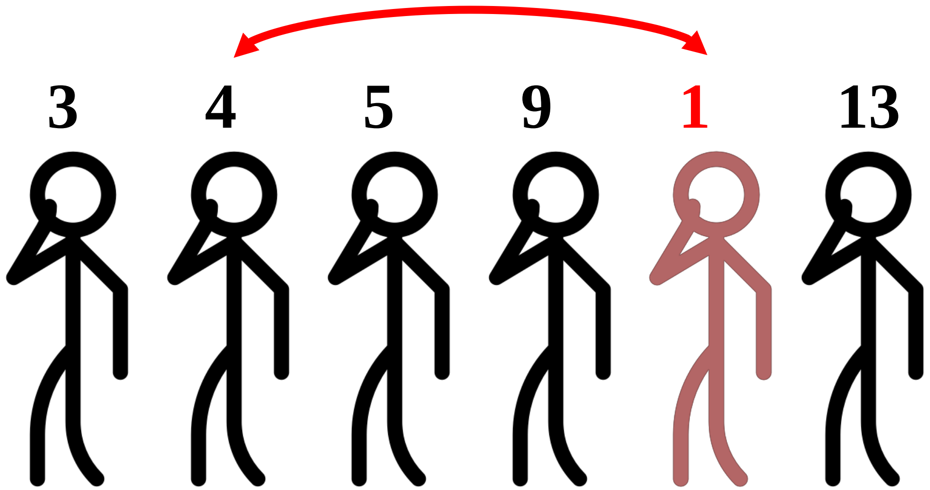


数据结构：优先队列

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
  - $Q.Insert()$
  - $Q.ExtractMin()$
  - $Q.DecreaseKey()$



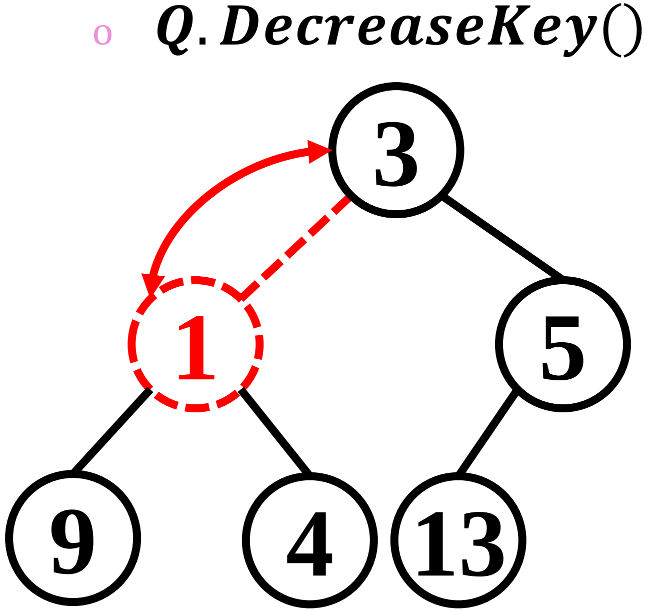


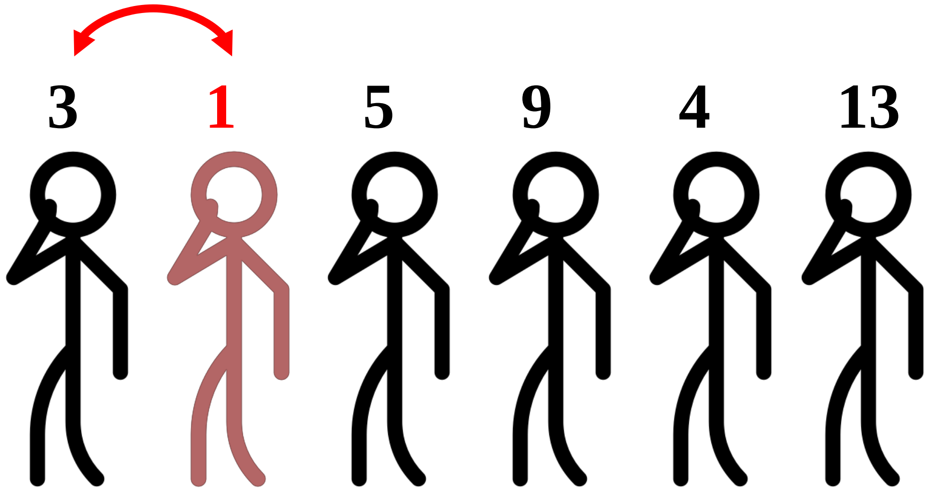
优先队列Q

数据结构：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
  - $Q.Insert()$
  - $Q.ExtractMin()$
  - $Q.DecreaseKey()$

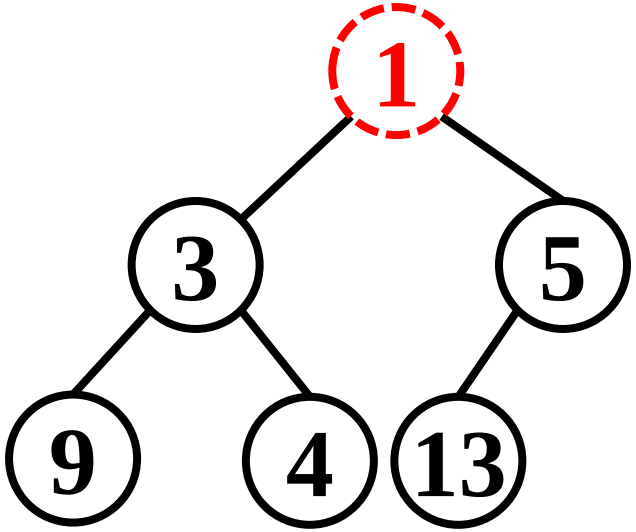


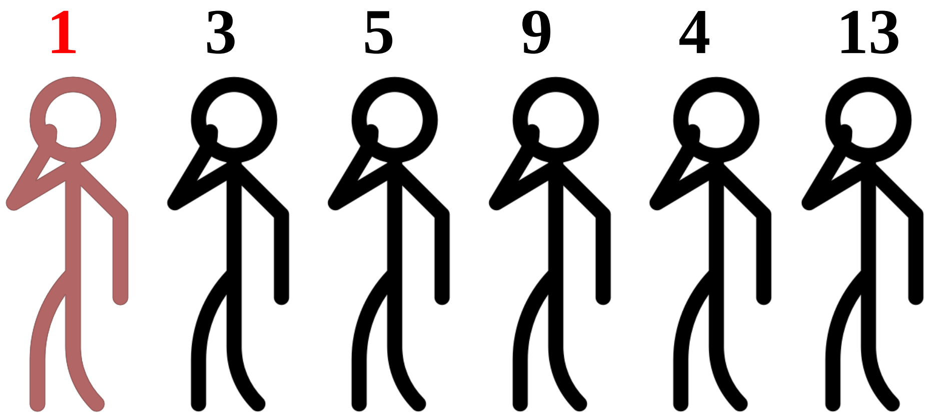


优先队列Q

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
    - $Q.Insert()$
    - $Q.ExtractMin()$
    - $Q.DecreaseKey()$



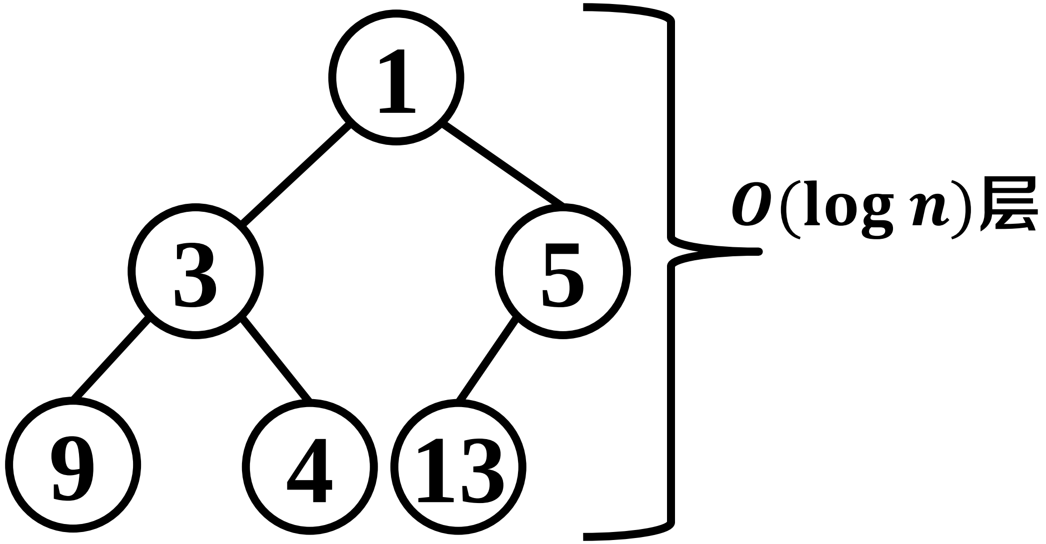


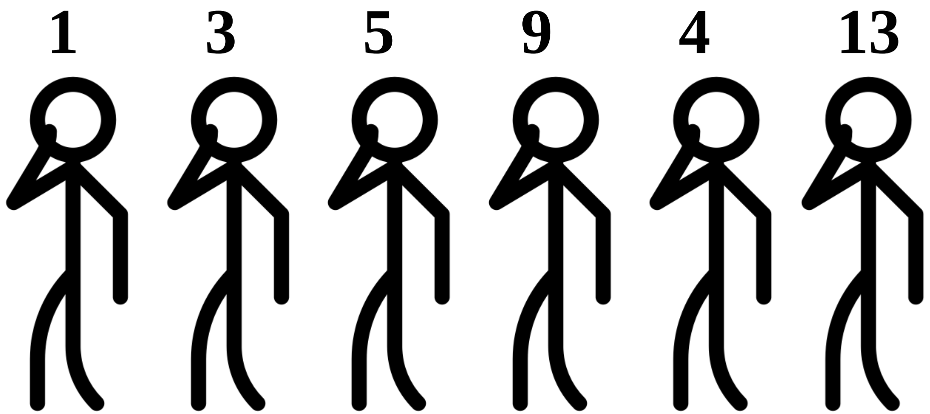
优先队列Q

数据结构：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
  - $Q.Insert()$
  - $Q.ExtractMin()$
  - $Q.DecreaseKey()$



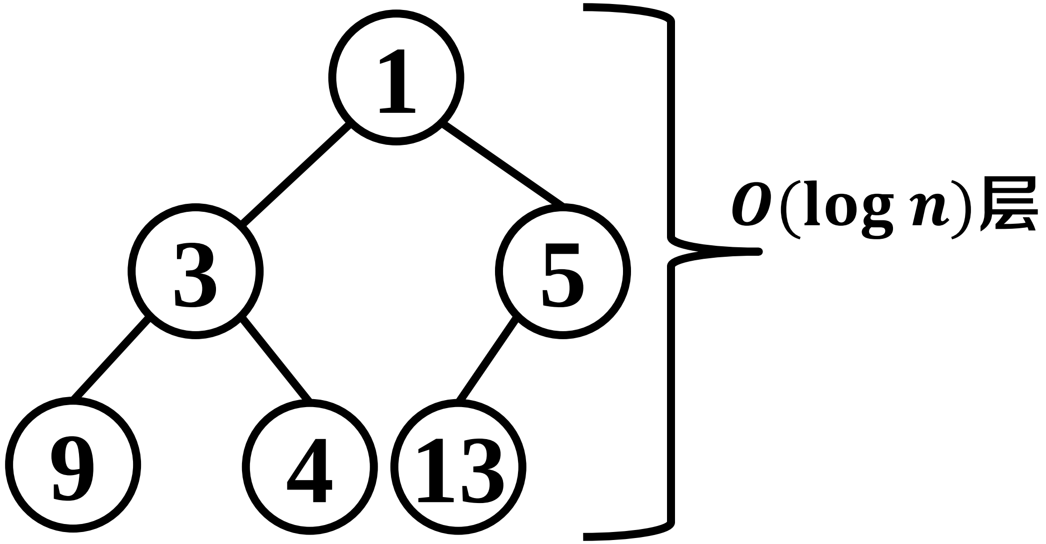


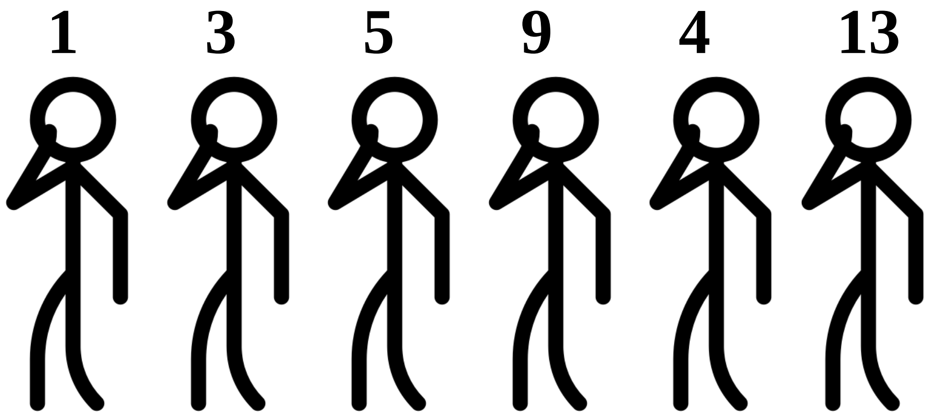
优先队列Q

数据结构：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
  - $Q.Insert()$ 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$O(\log n)$

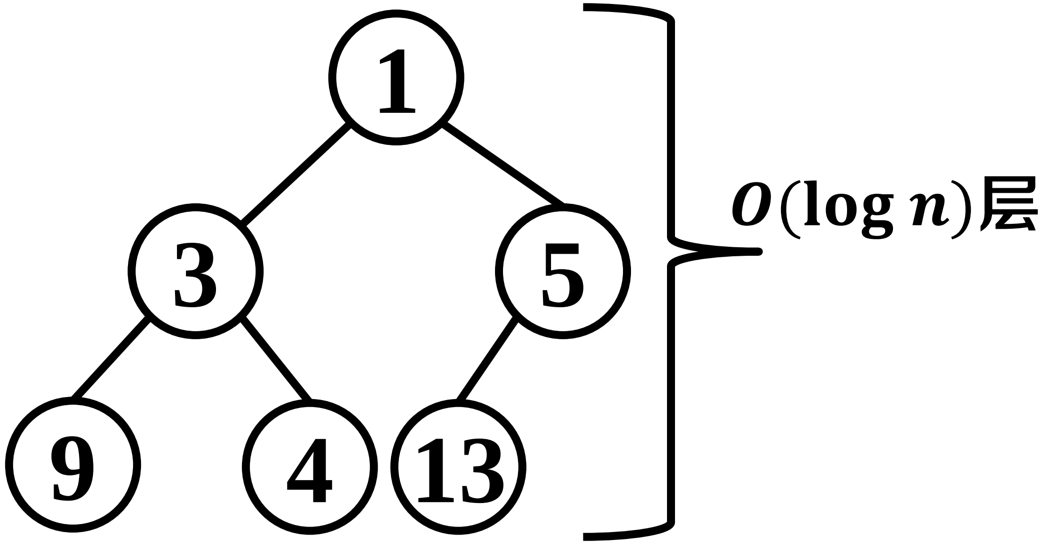


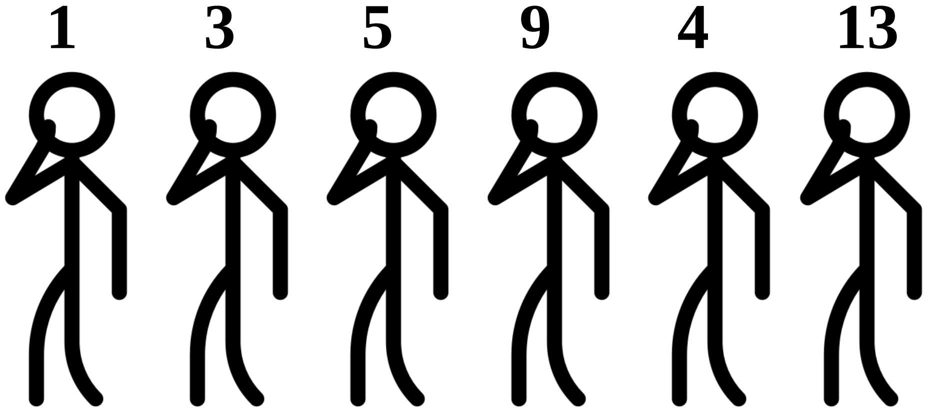


优先队列Q

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

伪代码

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

输入: 图$G =< V, E, W >$
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
$dist[1] \leftarrow 0$
$Q.Insert(V, dist)$

初始化辅助数组

伪代码

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

输入: 图$G=<V,E,W>$
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

伪代码

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

输入: 图$G=<V,E,W>$
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

伪代码

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

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

依次添加其他顶点

伪代码

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

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

选择安全边

伪代码

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

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

更新距离数组，调整优先队列

伪代码

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

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

标记顶点处理完成

复杂度分析

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

输入: 图$G=<V,E,W>$
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
$dist[1] \leftarrow 0$
$Q.Insert(V, dist)$

$O(|V|)$

复杂度分析

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

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

复杂度分析

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

```
//执行最小生成树算法
while 优先队列Q非空 do
    v ← Q.ExtractMin()                                  O(log|V|)
    for u ∈ G.Adj[v] do
        if color[u] = WHITE and w(v,u) < dist[u] then
            dist[u] ← w(v,u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))                 O(log|V|)
        end
    end                                                  O(deg(u)log|V|)
    color[v] ← BLACK
end
```

复杂度分析

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

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

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

$O(\log|V|)$

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

$O(\log|V|)$

复杂度分析

- **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

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

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

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

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

复杂度分析

- **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

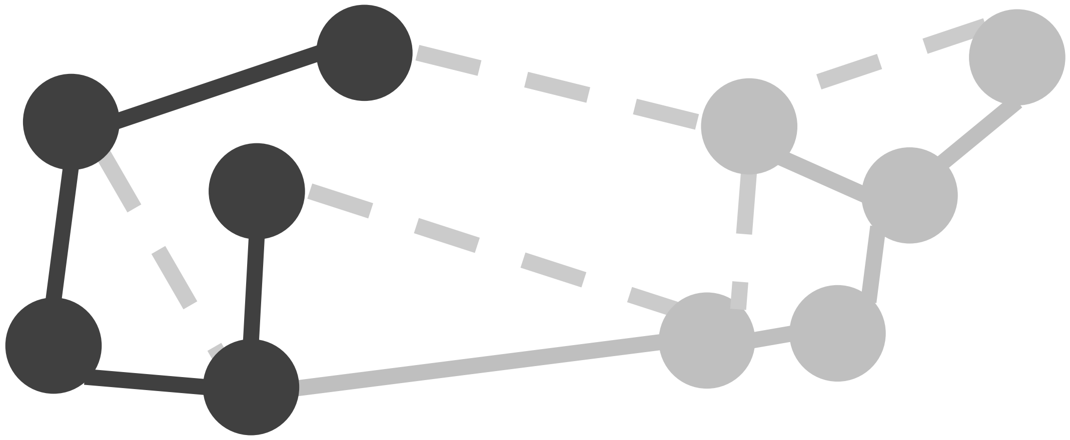
  $color[v] \leftarrow BLACK$

end

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

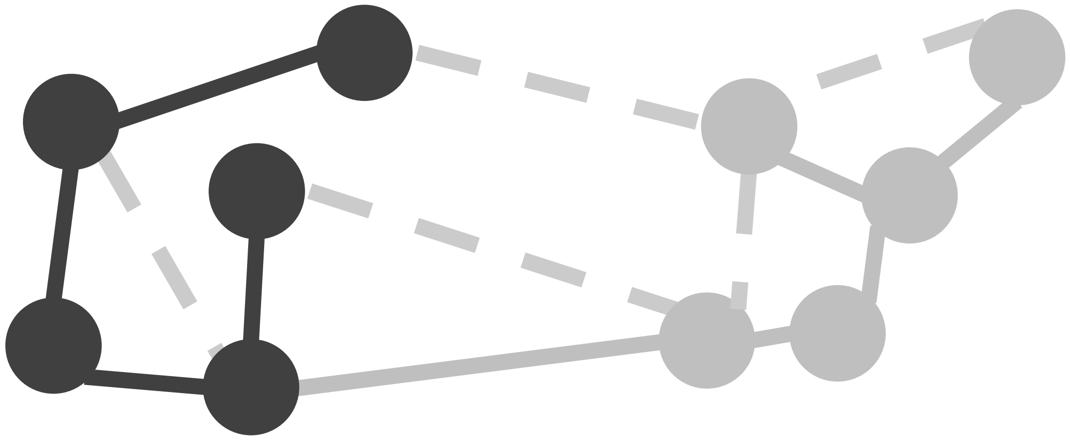
小结

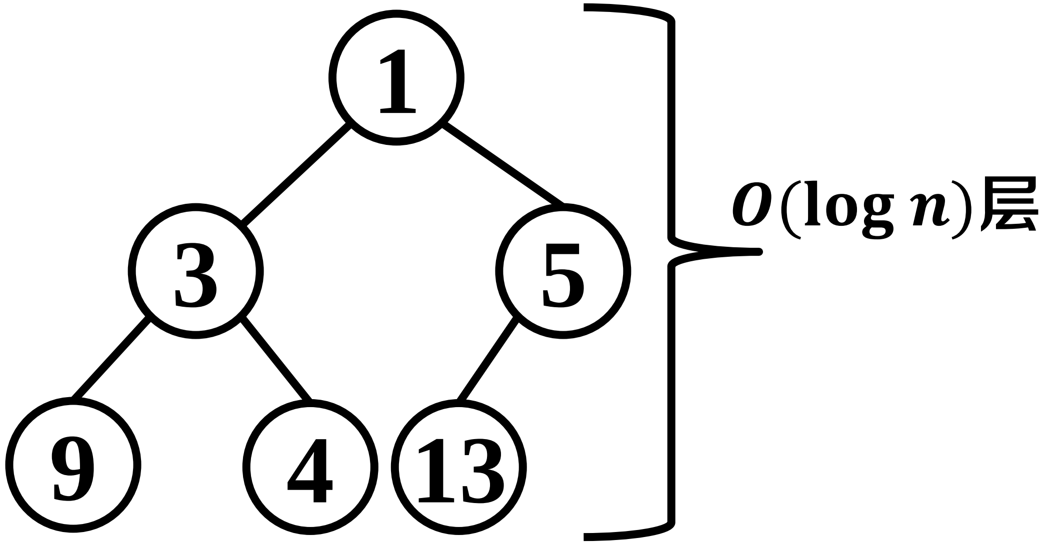
通用框架	Prim算法
判断是否成环	保持树的结构



小结

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





谢谢

BDA

Beihang University