算法设计与分析

Computer Algorithm Design & Analysis

2020.10

吕志鹏

zhipeng.lv@hust.edu.cn

群名称：算法设计与分析

群　号： 271069522

群名称：算法设计与分析2020

群　号：271069522

Chapter 23
Minimum Spanning Trees

最小生成树

布线问题:

在电子电路设计中，通常需要将多个组件的针脚连接在一起。设有n个针脚，则至少需要n-1根连线连接（每根连线连接两个针脚）。问**怎么连线才能使所使用的连线总长度最短**?

建模：最小生成树

将布线问题用一个连通无向图G=(V,E)表示，**结点表示针脚，边表示针脚之间的连线**。对每条边(u,v)∈E赋予**权重ω(u,v)**表示连接针脚（结点）u和v的代价（连线长度）。

问题的解:

找G中的一个**无环子集**$T \subseteq E$，使之既能够将所有的结点（针脚）连接起来，又具有最小的权重，即使得 $\omega(T)=\sum_{(u,v)\in T}\omega(u,v)$ 的值最小。

生成树：

由于T无环，并且连通所有的结点，所以T必然是一棵树，称这样的树为图G的生成树（Spanning Tree）。

- 图G的生成树是G的一个子图T=(V,E')，T是树且$V_T=V_G$，E'⊆E
- 对于带权图，生成树的成本等于树中所有边的权重之和。

最小生成树：具有最小权重的生成树称为最小成本生成树，简称最小生成树。

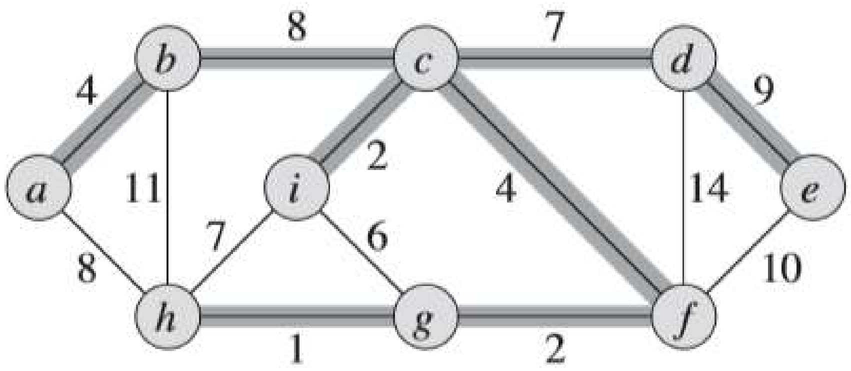


图23.1 连通图的最小生成树

- 一个连通图的最小生成树。
- 边上标记了权重，属于最小生成树的边用阴影表示。
- 生成树的总权重是37。
- 注：最小生成树并不一定唯一。

23.1 最小生成树的形成

对无向连通图G=(V,E)和权重函数ω:E→R，如何找出G的最小生成树？

- MST性质
- 一个贪心策略设计如下：

在每个时刻，该方法生长最小生成树的一条边，并在整个策略的实施过程中，管理一个遵守下述循环不变式的边的集合A：

在每遍循环之前，A是某棵最小生成树的一个子集。

- **处理策略**：每一步，我们选择一条边(u,v)加入集合A，使得A不违反循环不变式，即A∪{(u,v)}后还是某棵最小生成树的子集。

- 这样的边使得我们可以“安全地”将之加入到集合A而不会破坏A的循环不变式，因此称之为集合A的“安全边”。

算法描述：

GENERIC-MST(G, w)

```
1  A = ∅
2  while A does not form a spanning tree
3      find an edge (u, v) that is safe for A
4      A = A ∪ {(u, v)}
5  return A
```

循环不变式：

初始化：在算法第一行之后，集合A为空，直接满足循环不变式。

保持：算法2~4行的循环通过只加入安全边来构造A，故可以维持循环不变式。

终止：所有加入到A中的边都属于某棵最小生成树，因此，某个时刻while一定终止，且第5行所返回的集合A必然是一棵最小生成树。

说明：算法第3行找一条安全边，这条安全边必然是存在的。因为在执行算法第3行时，循环不变式告诉了我们存在一棵生成树，满足 $A \subseteq T$ 。在进入while循环时，**A是T的真子集**，因此必然存在一条边 $(u, v) \in T$ ，使得 $(u, v) \notin A$ ，并且(u, v)对于集合A是安全的。

怎么寻找安全边?

定义:

切割：无向图G=(V,E)的一个切割(S,V-S)是集合V的一个划分。

如图所示：

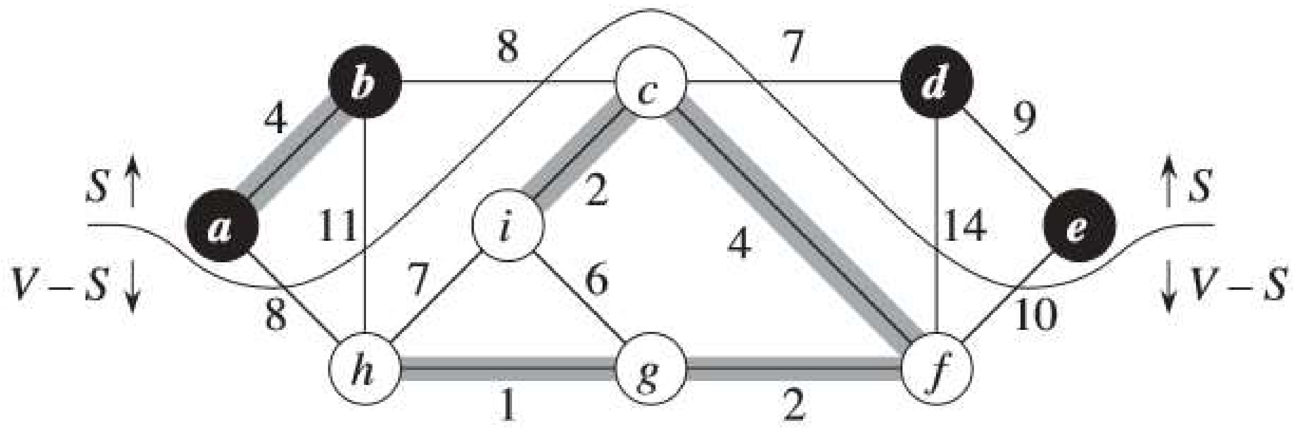
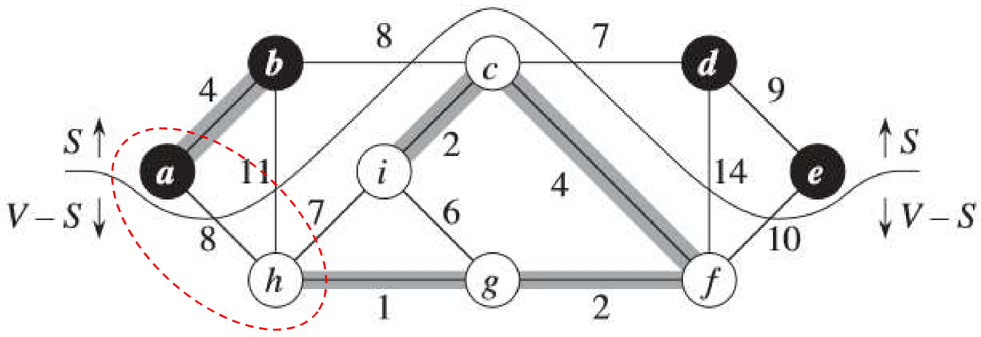


图23-2 图23-1的一个切割(S,V-S)



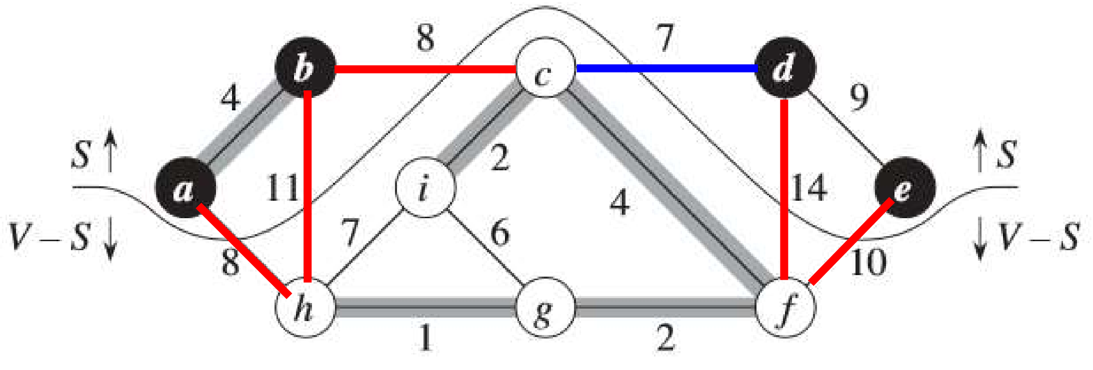
横跨切割：如果一条边(u,v)∈E的一个端点在集合S中，另一个端点在集合V-S中，则称该条边**横跨切割**(S,V-S)。

尊　　重：如果**边集A**中不存在横跨该切割的边，则称该切割**尊重**集合A。

轻量级边：在横跨一个切割的所有边中，权重最小的边称为**轻量级边**。

- 轻量级边可能不是唯一的。
- 一般，如果一条边是满足某个性质的所有边中权重最小的，则称该边是满足给定性质的一条轻量级边。

例



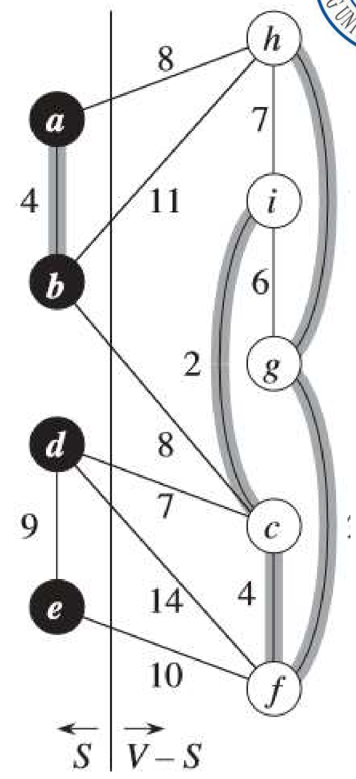


图23-1的一个切割(S,V-S)

- **横跨切割(S, V−S)的边**：

  (b, c), (c, d), (b, h), (d, f), (a, h), (e, f)

- **轻量级边**：(c, d)是唯一一条轻量级边，权重为7。

- **尊　　重**：

  将图中加了阴影的边，(a, b)、(c, i)、(c, f)、(f, g)、(g, h)构成一个集合A，其中不存在横跨该切割的边，故切割(S, V−S)尊重集合A。

- 将切割(S, V−S)中两个集合的结点分别画在左、右两边，左边是集合S中的结点，右边是集合V−S中的结点。
- 横跨切割的边一端连接左边的一个结点，一端连接右边的一个结点。

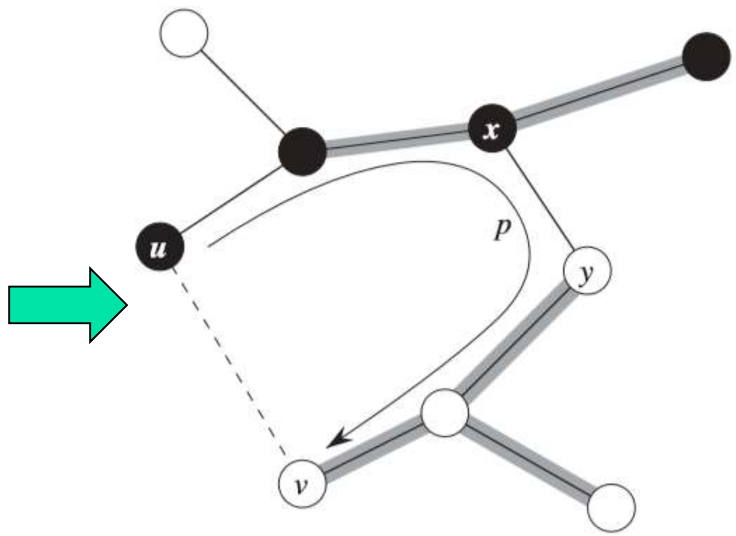
选择安全边的规则

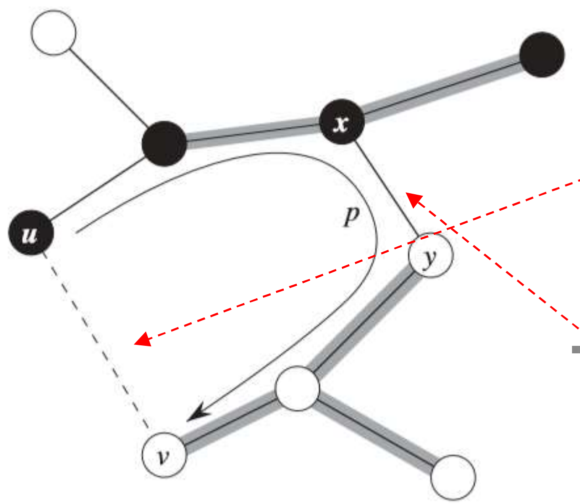
定理23.1 设G=(V, E)是一个在边E上定义了实数值权重函数ω的连通无向图。设集合A为E的一个子集，**且A包含在图G的某棵最小生成树中**，设(S, V-S)是图G中尊重集合A的任意一个切割，又设**(u,v)是横跨切割(S,V-S)的一条轻量级边**。那么边(u, v)对于集合A是安全的（即MST性质）。

证明：设T是一棵包含A的最小生成树，且T不包含轻量级边(u,v)

- 注：若T包含轻量级边(u,v)，则证毕。

T中包含有G的所有结点，且是一棵树，所以(u,v)与T中从结点u到结点v的简单路径p形成一个环。





对于(u, v)而言，

- u和v分别处在它所横跨的切割(S, V-S)的两端（如图所示，所有黑色的结点位于集合S中，所有白色的结点位于集合V-S中）；
- 且T中至少有一条属于p的边也横跨该切割（如图中的(x, y)所示）。

设(x,y)是T中属于简单路径p但横跨该切割的边，且根据已知条件：切割（S, V-S）尊重集合A，所以边(x,y)不在集合A中。

由于边(x, y)位于树T中，是从u到v的唯一简单路径上的一条边，所以将该边删除会导致T被分解为两个连通分量。

将(u,v)加上去，则可以将这两个连通分量连接起来再次形成一棵新的生成树，记为T’=T-{(x, y)}∪{(u, v)}。

由于边(u, v)是横跨切割(S, V−S)的一条轻量级边，而且边(x, y)也横跨该切割，所以应有ω(u, v)≤ ω(x, y)。因此，

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

而T是一棵最小生成树，所以还应有ω(T)≤ ω(T’)。

所以ω(T)=ω(T’)，即**T'一定也是一棵最小生成树**。

另，因为 $A \subseteq T$ ，且(x, y) $\notin A$ ，所以 $A \subseteq T'$ 。因此

$$A \cup \{(u, v)\} \subseteq T'$$

由于T’是最小生成树，所以(u, v)对于集合A是安全的。

证毕。

循环不变式：在每遍循环之前，A是某棵最小生成树的一个子集。

这样的边使得我们可以“安全地”将之加入到集合A而不会破坏A的循环不变式，因此称之为集合A的“安全边”

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

基于定理23.1理解算法GENERIC-MST

- 在算法推进的过程中，**集合A始终保持无环状态。**
  - 因为加入A中每条边都是安全的，使A始终保持为一棵最小生成树子集的状态。
- 算法执行的任意时刻，**图G_A=(V,A)是一个森林。**
  - G_A中的每个连通分量是一棵树
    - 某些连通分量可能是仅含一个结点的树，如在初始时，A=Ø，G_A中有|V|棵树，每棵树都只有一个结点。
  - 对于安全边(u, v)，由于A∪{(u, v)}必须无环，所以（u, v）连接的是G_A中的两个不同连通分量。

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

- while循环执行|V|-1次，每次找出构造最小生成树所需的一条边。
  - 初始时，A=Ø，G_A中有|V|棵树；
  - 其后每遍循环将G_A中树的数量减少1。
  - 当整个森林只含有一棵树时，算法终止。此时A是问题的解(最小成本生成树)。

推论 23.2 设G=(V,E)是一个无向连通图，并有定义在边集合E上的实数值权重函数ω。设集合A为E的一个子集，且A包含在图G的某棵最小生成树中。设$C=(V_C, E_C)$为森林$G_A=(V,A)$中的一个连通分量，边$(u,v)\in E$、$(u,v)\notin A$ 是C连接和G_A中其它连通分量的所有边中权重最小的边。则边(u,v)对于集合A是安全的。

证明：由于C是一个连通分量，与其它连通分量没有边连接，所以定义在C的结点集V_C上的切割$(V_C, V-V_C)$尊重集合A，即A中没有横跨该切割的边。而边(u,v)就是横跨该切割的一条轻量级边，根据定理23.1，(u,v)对于集合A是安全的。

证毕

23.2 Kruskal和Prim算法

Kruskal和Prim算法是求解最小生成树的两个经典算法。它们都是GENERIC-MST算法的具体细化，每种算法都使用一条具体的规则来确定GENERIC-MST算法第3行所描述的安全边：

- **Kruskal算法**：集合A始终是一个森林，开始时，其结点集就是G的结点集，并且A是所有单节点树构成的森林。之后每次加入到集合A中的安全边是G中连接A的两个不同分量的权重最小的边。
- **Prim算法**：集合A始终是一棵树，每次加入到A中的安全边是连接A和A之外某个结点的边中权重最小的边。

注：Kruskal算法和Prim算法都是典型的**贪心算法。**

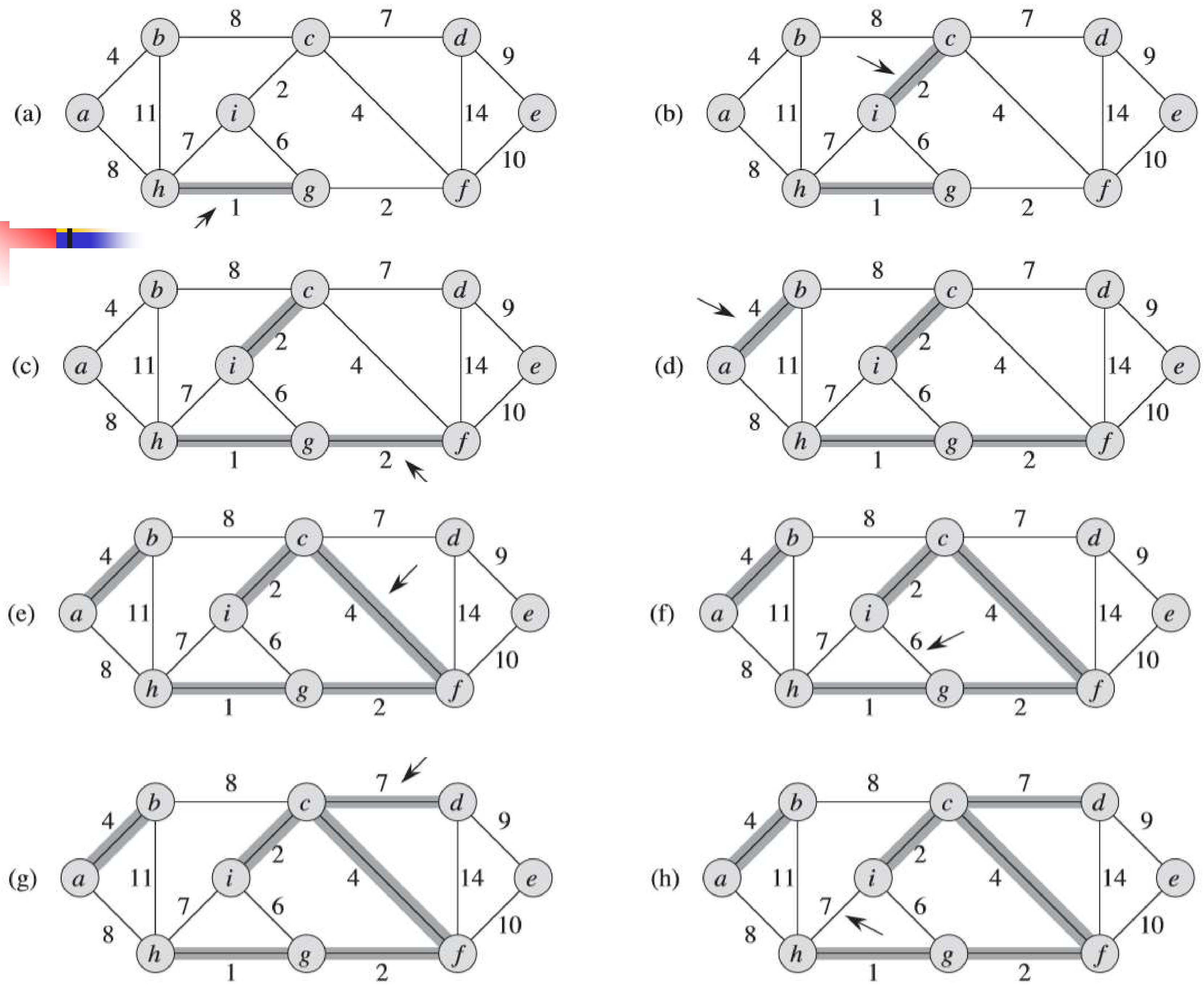
1. Kruskal算法

Kruskal算法找安全边的方法：在所有连接森林中两棵不同树的边中，找权重最小的边(u, v)。

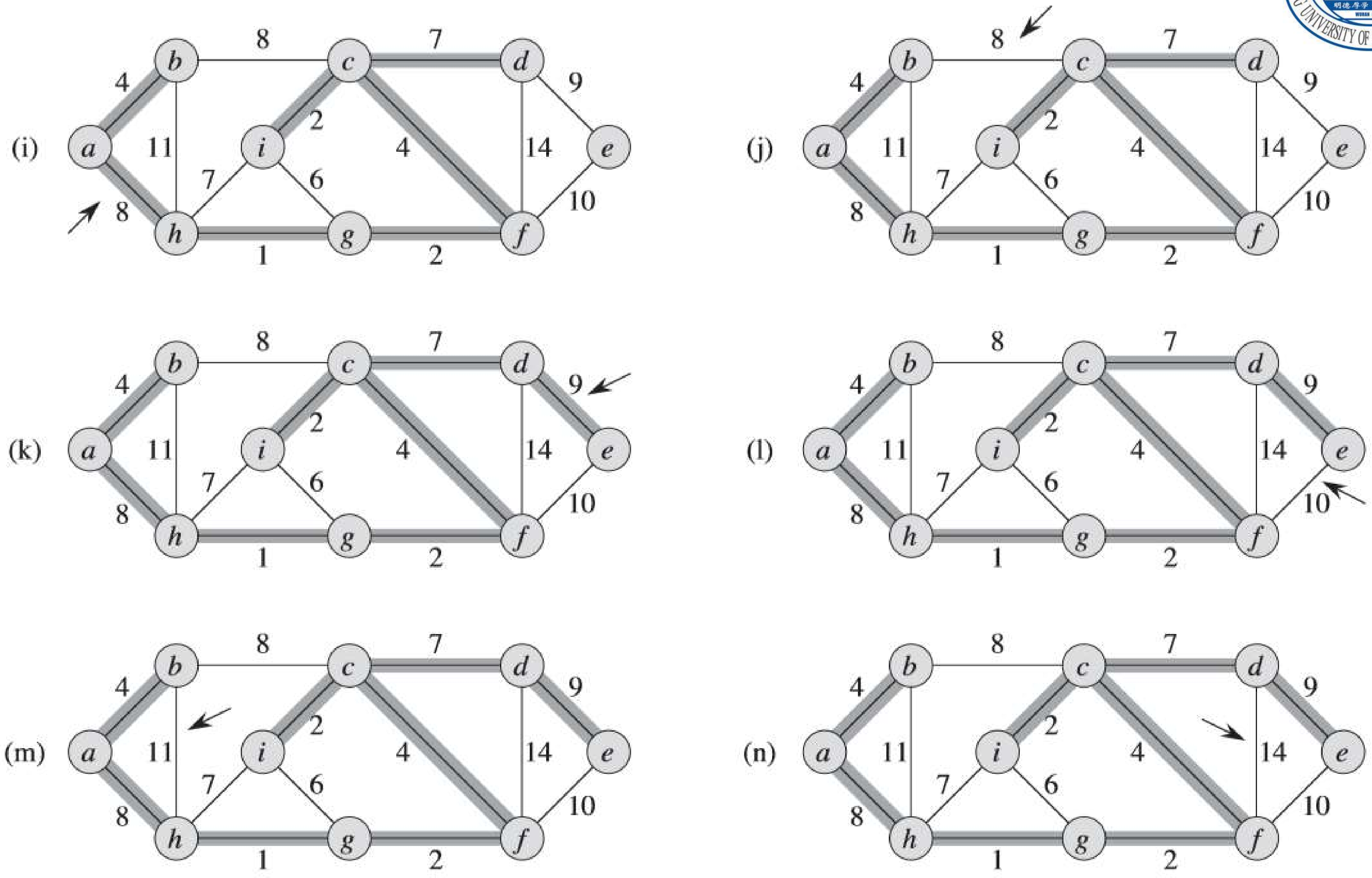
- 设C_1和C_2是边(u, v)所连接的两棵树，则边(u, v)一定是C_1连接其它连通分量（包括树C_2）的一条轻量级边，根据推论23.2，边(u, v)是C_1的一条安全边。

```
MST-KRUSKAL(G, w)
A = Ø
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

借助不相交集合数据结构实现



在图23-1上执行Kruskal算法。加了阴影的边属于不断增长的森林A。



在图23-1上执行Kruskal算法。加了阴影的边属于不断增长的森林A。
（continue）

Kruskal算法的时间

- Kruskal算法的运行时间依赖于**不相交集合**数据结构的具体实现。
- Kruskal算法的时间为：$O(E\ \lg E)$。
  - 如果再注意到$|E|<|V|^2$，则有$\lg|E|=O(\lg V)$，所以Kruskal算法的时间可表示为$O(E\ \lg V)$。
- （见教材P366）

2. Prim算法

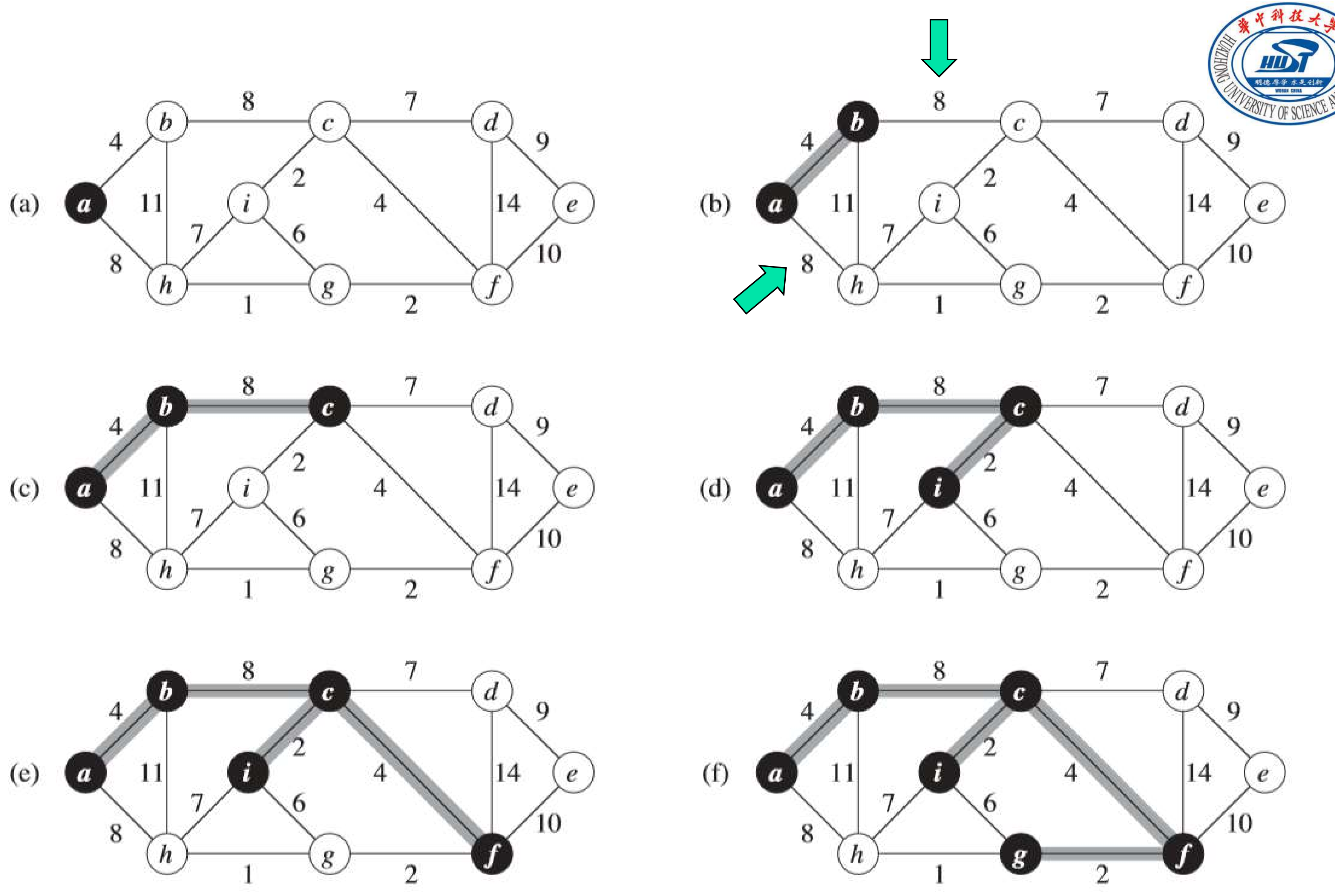
Prim算法的每一步是在**连接集合A和A之外所有结点**的边中，选择一条轻量级边加入到A中。根据推论23.2，这条规则所加入的边对于A也是安全的。

Prim算法的基本性质：集合A中的边总是构成一棵树。

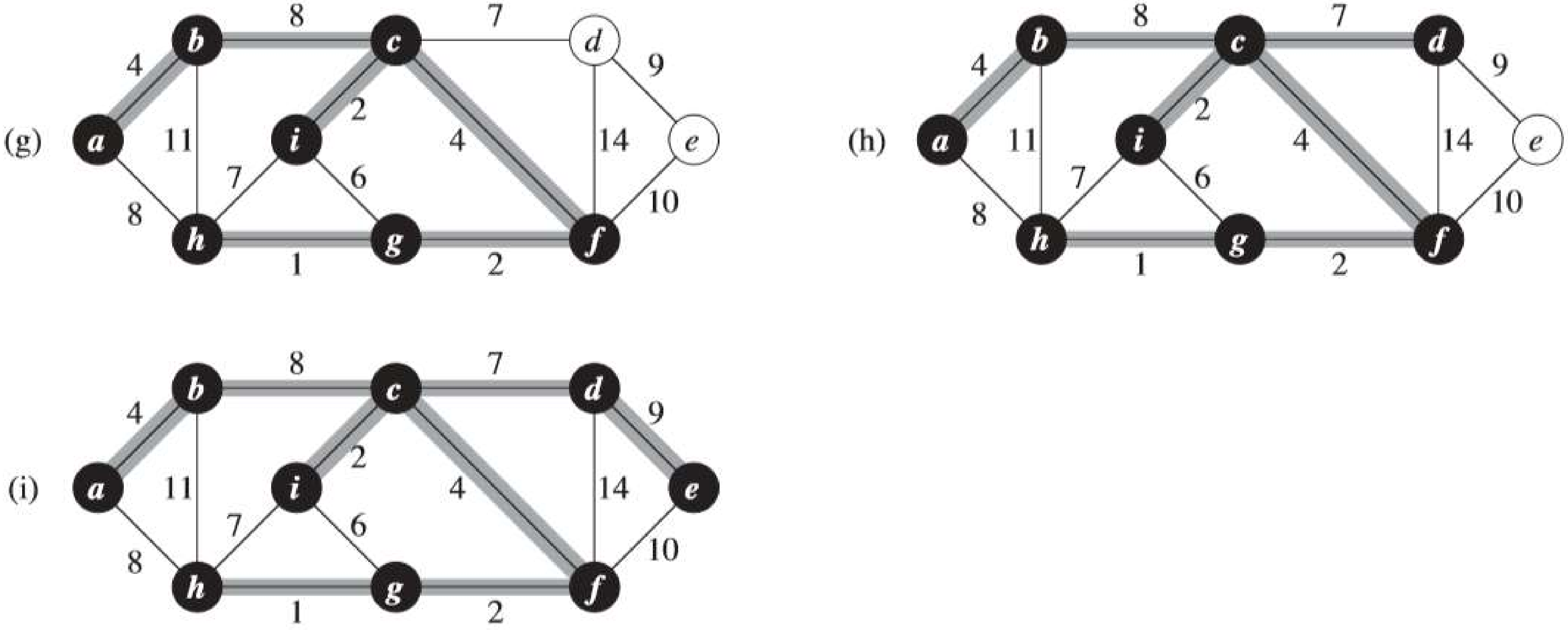
```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ Ø
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

- r是树根，从r.key=0并首次选中r开始。
- 结点的属性key的值是其连接至A的最小权重。
- 这里使用最小优先队列Q，以快速地选择下一条边
- v.π记录结点v在树中的父结点。
- 算法终止时，最小优先队列Q为空，G的最小成本生成树为：

  $A=\{(v,v.\pi):v\in V-\{r\}\}$



在图23-1上执行Prim算法，初始时根结点为a。加阴影的边和黑色的结点都属于树A。
在算法的每一步，树中的结点就决定了图的一个切割。横跨该切割的一条轻量级边被加入树中。



在图23-1上执行Prim算法，初始时根结点为a。加阴影的边和黑色的结点都属于树A。在算法的每一步，树中的结点就决定了图的一个切割。横跨该切割的一条轻量级边被加入树中。（continue）

Prim算法的时间

- Prim算法的运行时间依赖于最小优先队列Q的具体实现。
  - 可用二叉最小优先队列的方式实现。
  - 每次EXTRACT-MIN的时间是O($\lg V$)。
  - EXTRACT-MIN的总时间是O($V\lg V$)。
- 其它时间：第11行的赋值操作共需O($E\lg V$)。

Prim算法的时间为：$O(V\lg V+E\lg V)=O(E\lg V)$。

  - 从渐进意义上看，Kruskal和Prim算法具有相同的运行时间。

- (见教材P369)