6.图

(g) Prim算法

邓俊辉

deng@tsinghua.edu.cn

最小 + 支撑 + 树

❖ 连通网络N = (V; E)的子图T = (V; F)

1) 支撑/spanning

覆盖N中所有顶点

2) 树/tree

连通且无环，$|V| = |F| + 1$

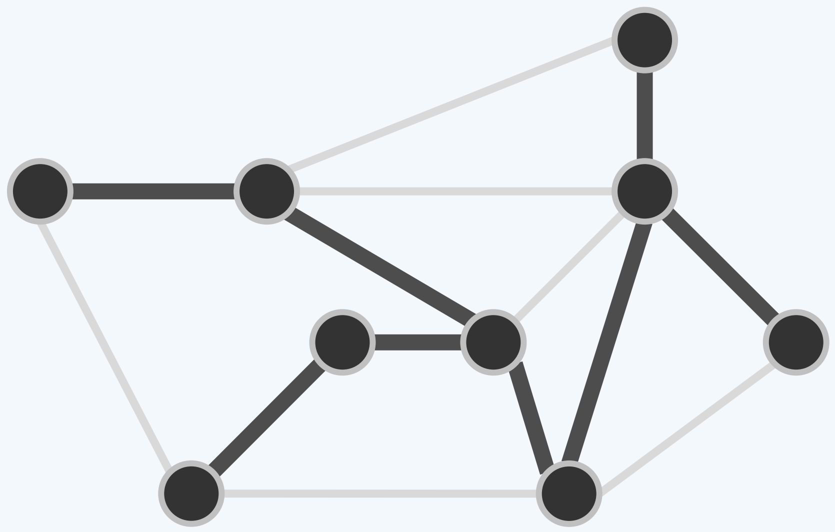
加边出单环，再删同环边即恢复为树

删边不连通，再加联接边即恢复为树

不难验证，同一网络的支撑树不唯一

3) 最小/minimum

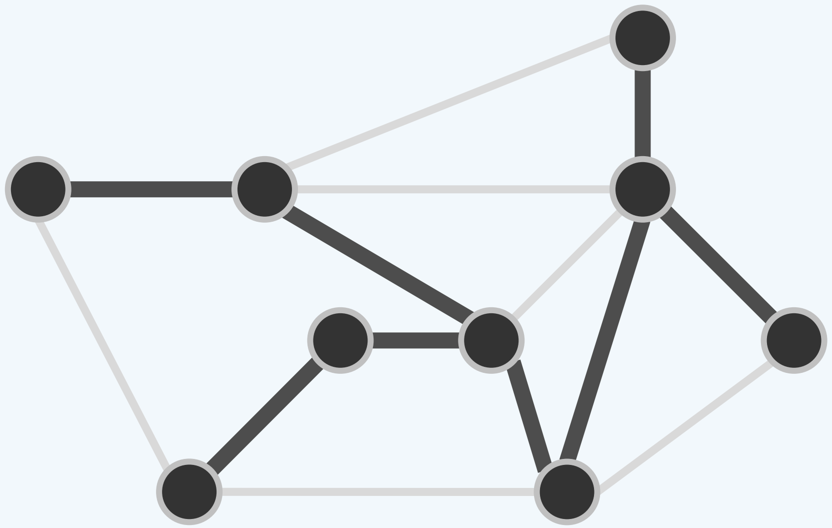
各边总权重$wt(T) = \sum_{e \in F} wt(e)$达到最小



MST

❖ 谁感兴趣？

电信公司、网络设计师、VLSI布线算法设计师、...



❖ 为何重要？

应用中常见的共性问题，也是很多优化问题的基本**模型**

自身可**有效**计算 //具体算法稍后介绍

为许多NP问题提供足够好的**近似**解 //比如，Euclidean TSP

❖ 算法

Boruvka-1926, Jarnik-1930/Prim-1956, Kruskal-1956, ...

Karger-Klein-Tarjan-1995, Chazelle-2000

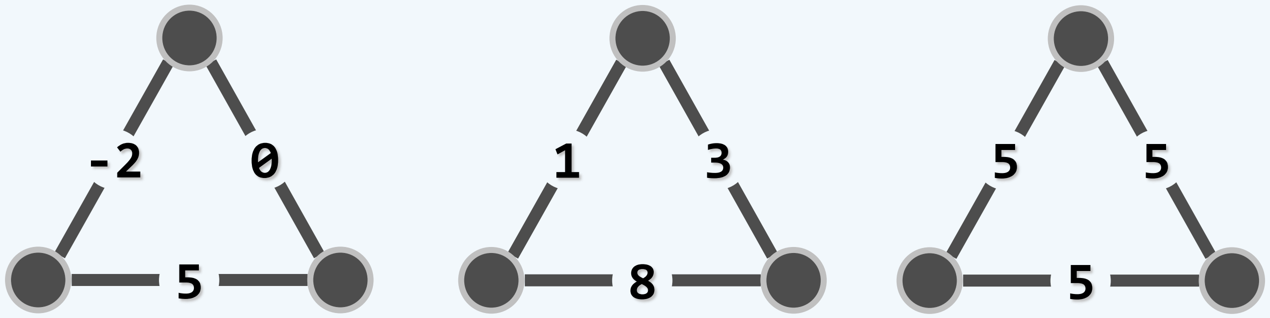
...是否存在$\mathcal{O}(n + e)$算法？

退化

❖ 权值必须是正数？

允许为零，会有什么影响？

允许为负数呢？



❖ 所有支撑树所含的边数，必然相等

故可统一调整：increase(1 - findMin())

❖ The minimum？

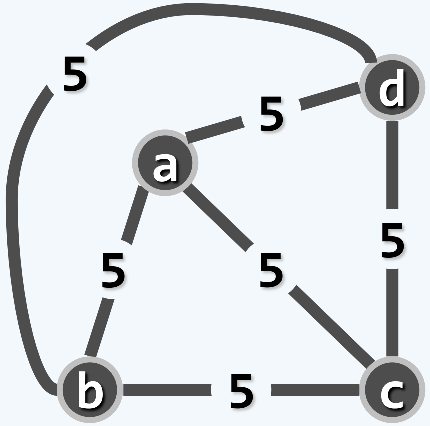
A minimal　同一网络N可能有多棵MST

The minimum！可强制消除歧义...

❖ 合成数（composite number）

(w(u, v), min(u, v), max(u, v))

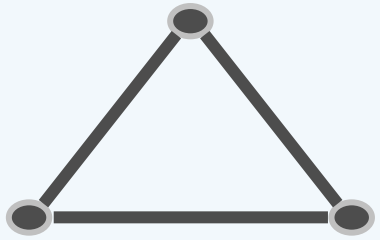
5ab < 5ac < 5ad < 5bc < 5bd < 5cd



蛮力算法

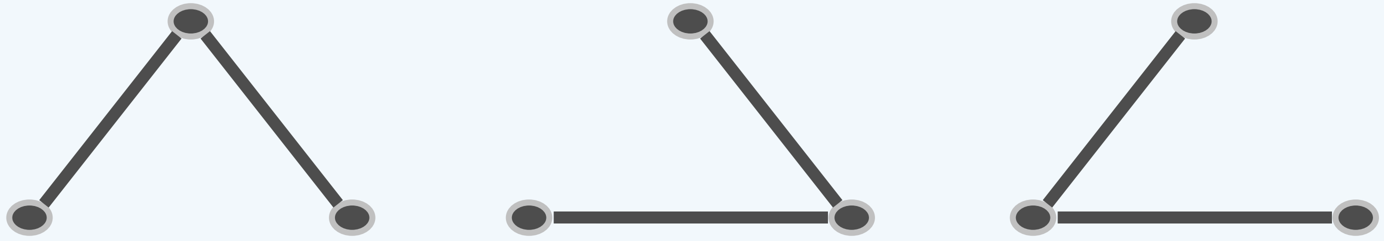
❖ 枚举出N的所有支撑树，从中找出代价最小者

❖ 这一策略是否可行，取决于...



❖ n个互异顶点组成的图，可能有多少棵支撑树？

n = 1	1
n = 2	1
n = 3	3
n = 4	16
...	...



❖ Cayley公式： 联接n个互异顶点的树共有n^{n-2}棵，或等价地，

完全图K_n有n^{n-2}棵支撑树

❖ 如何高效地构造MST呢？

割 & 极短跨边

❖ 设(U; V\U)是N的割(cut)

❖ 【Cut Property-A】

若：(u, v)是该割的**极短**跨边(shortest crossing edge)

则：必**存在**一棵包含(u, v)的MST

❖ 反证：假设(u, v)未被**任何**MST采用...

❖ **任取**一棵MST，将(u, v)加入其中，于是

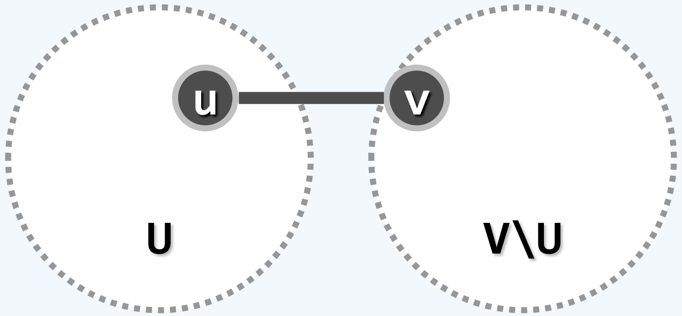
将出现**唯一**的回路，且该回路必经过

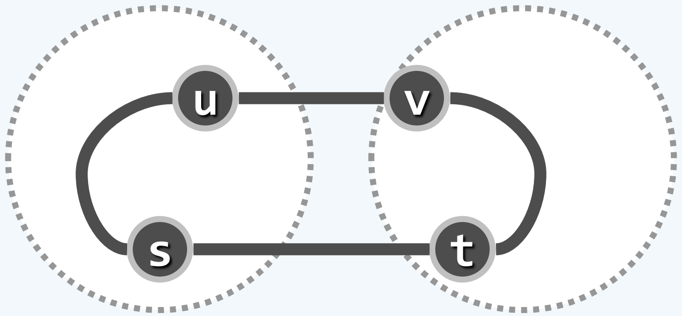
(u, v)以及**至少**另一跨边(s, t)

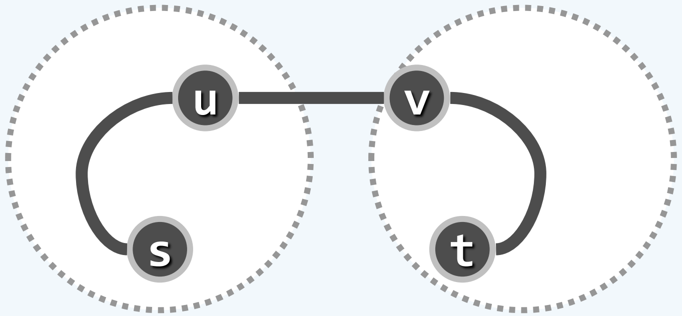
❖ 现在，将原MST中的(s, t)替换为(u, v)...

❖ 【Cut Property-B】

反之，N的**任一**MST也必通过极短跨边联接**每一**割





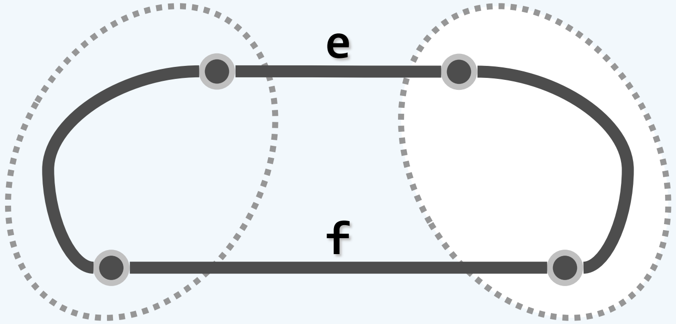


环 & 极长环边

❖ 设T是N的一棵MST，且在N中添加边e后得到N'

❖ 【Cycle Property】 若：沿着e在T中对应的环路，f为一**极长**边

则：T - {f} + {e}即为N'的一棵MST



❖ 1)若e为环路上的**最长**边，则与前同理，e不可能属于N'的MST

此时，f = e，T - {f} + {e} = T依然是N'的MST

❖ 2)否则有：|e| ≤ |f|；移除f后T - {f}一分为二，对应于N/N'的割

在N/N'中，f/e应是该割的**极短**跨边

此割在N和N'中导出的一对互补子图完全一致

故，这对子图各自的MST经e联接后，即是N'的一棵MST

算法

- 从$T_1 = (\{v_1\};\ \varnothing)$开始，逐步构造$T_2$、$T_3$、…、$T_n$，其中

  v_1可以任选

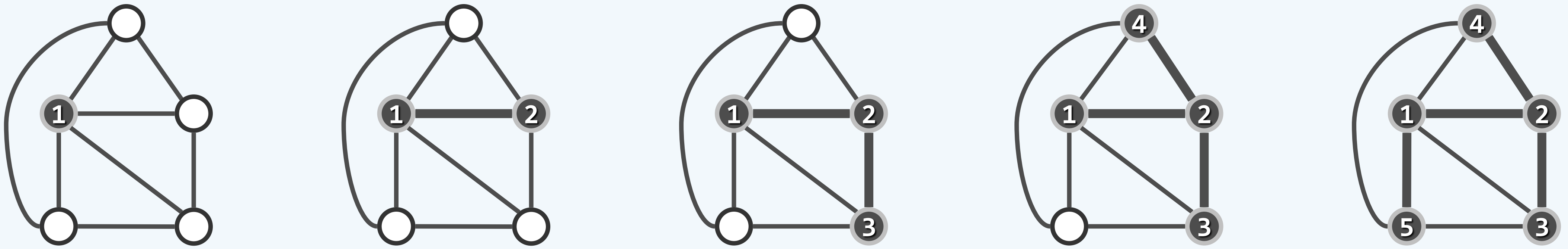
  $T_k = (V_k;\ E_k)$，$|V_k| = k$，$|E_k| = k-1$，$V_k \subset V_{k+1}$

- 由以上分析，为由T_k构造T_{k+1}，只需

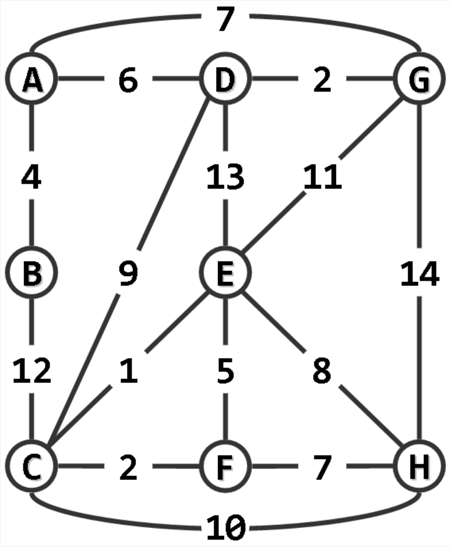
  将$(V_k : V \backslash V_k)$视作原图的一个割

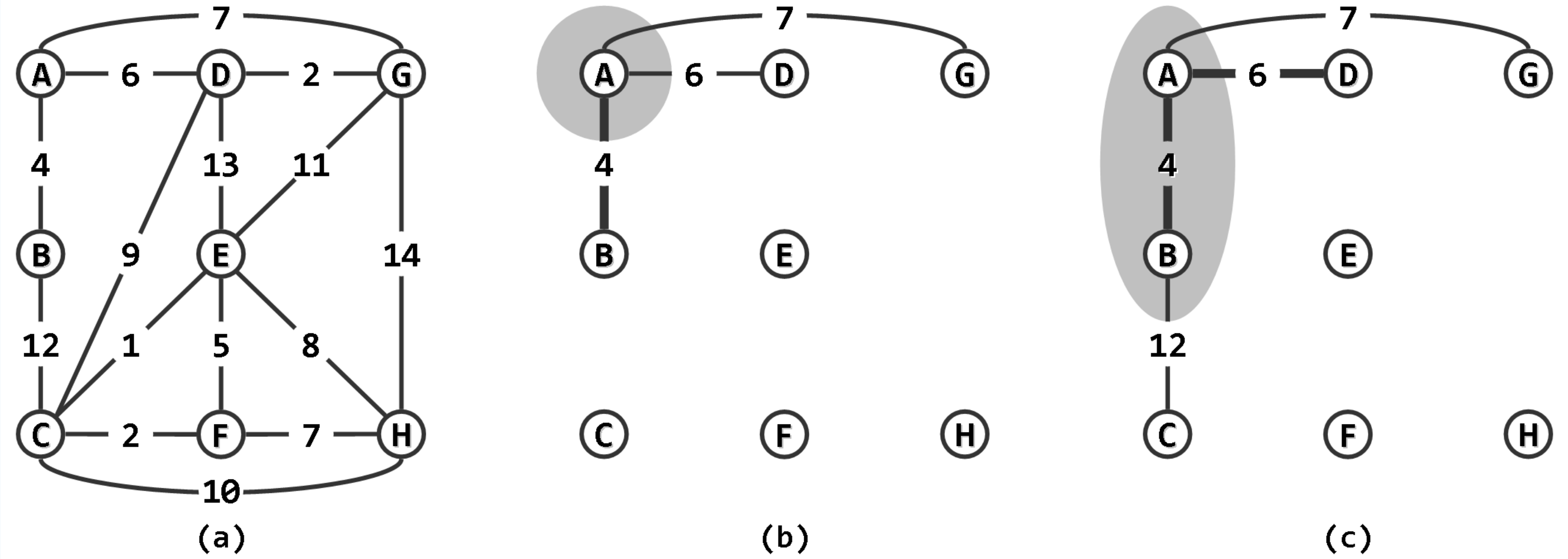
  在该割的所有跨边中，找出**极短**者$e_k = (v_k,\ u_k)$

  令$T_{k+1} = (V_{k+1};\ E_{k+1}) = (V_k \cup \{u_k\};\ E_k \cup \{e_k\})$

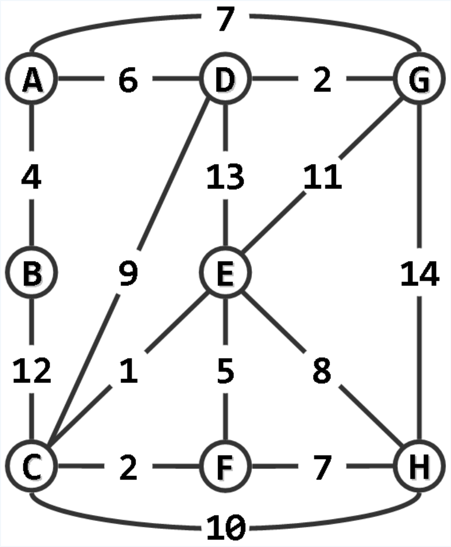


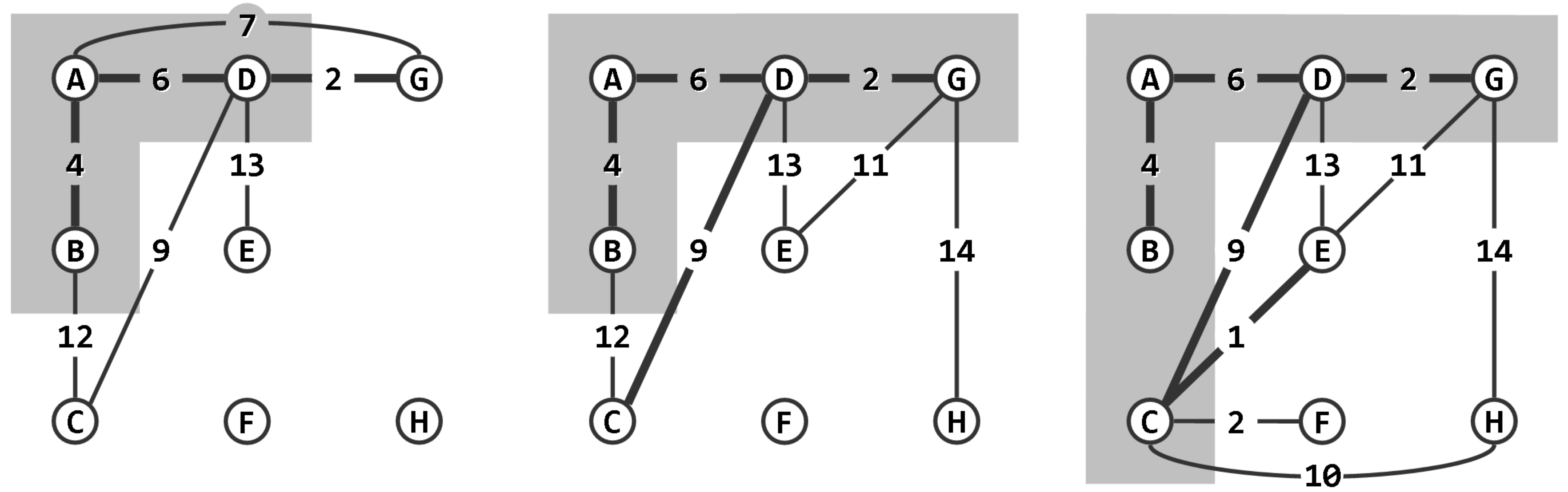
实例


7
A
6
D
2
G
4
13
11
B
9
E
14
12
1
5
8
C
2
F
7
H
10


7
A
6
D
2
G
4
13
11
B
9
E
14
12
1
5
8
C
2
F
7
H
10
(a)
7
A
6
D
G
4
B
E
C
F
H
(b)
7
A
6
D
G
4
B
E
12
C
F
H
(c)

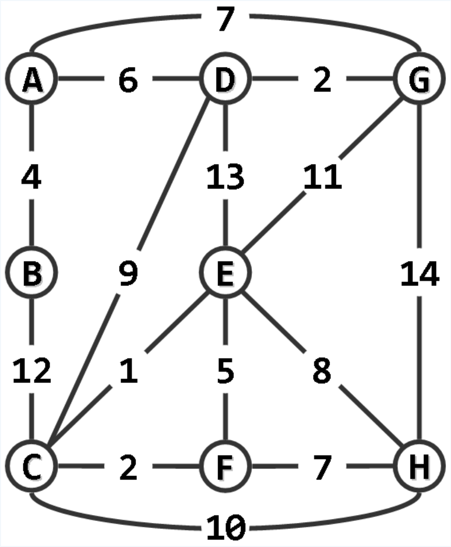
实例

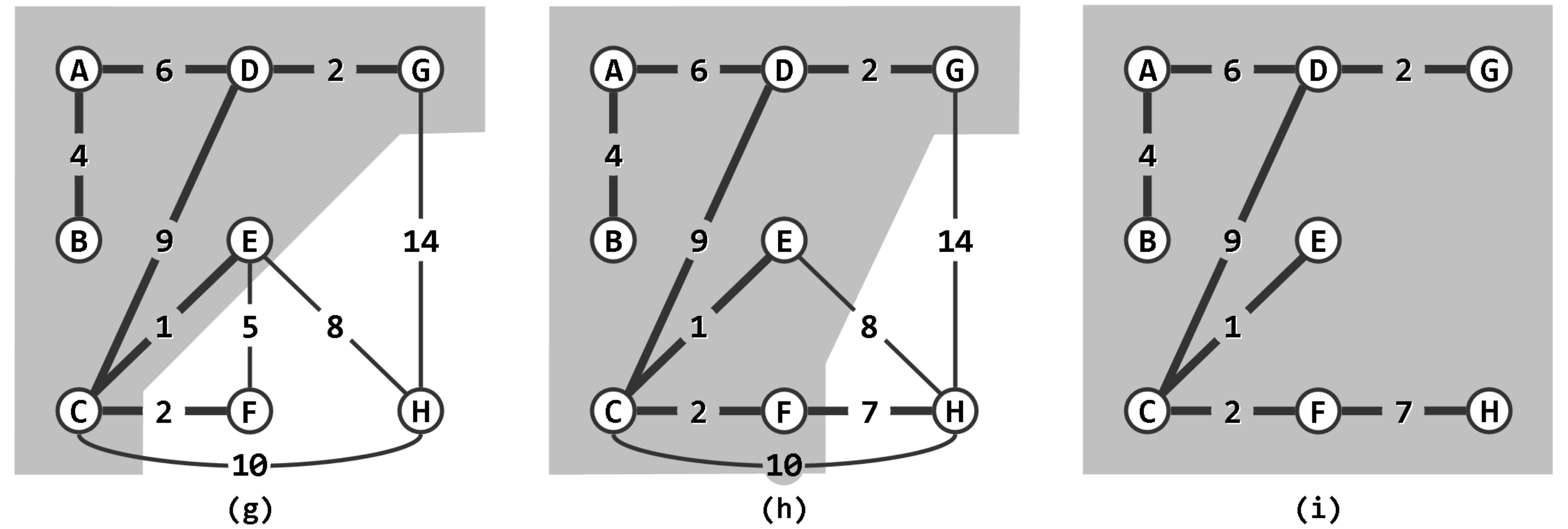

7
A 6 D 2 G
4 13 11
B 9 E 14
12 1 5 8
C 2 F 7 H
10


7
A 6 D 2 G
4 13
B 9 E
12
C F H
A 6 D 2 G
4 13 11
B 9 E 14
12
C F H
A 6 D 2 G
4 13 11
B 9 E 14
1
C 2 F H
10

(d) (e) (f)

实例





正确性

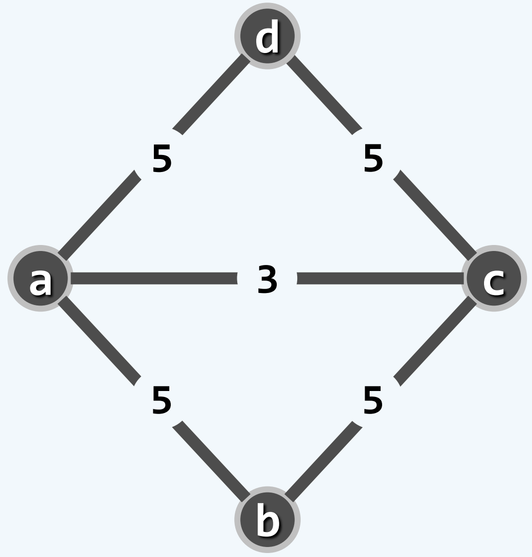
❖ 设Prim依次选取边{ e_2, e_3, ..., e_n }，构造出树T

则其中每一条边e_k都属于某棵MST

——的确如此，但在MST**不唯一**时并不充分

——满足这一性质的一组边，未必构成一棵MST

❖ 反例...



❖ 可行的证明方法...

1)在不增加总权重的前提下，可以将任一MST**转换**为T

数学归纳

2)每一T_k都是某棵MST的**子树**，$1 \le k \le n$

数学归纳

实现

- 对于V_k之外的每一顶点v，令：priority(v) = v到V_k的距离

  于是套用优先级遍历算法框架...

- 为将T_k扩充至T_{k+1}，可以

  选出优先级最高的（极短）跨边e_k及其对应顶点u_k，并将其加入T_k

  随后，更新V_{k+1}之外每一顶点的优先级（数）

- 注意，优先级数随后有可能改变（降低）的顶点，必与u_k邻接

- 因此，只需遍历枚举u_k的每一邻接顶点v，并取

  priority(v) = min(priority(v), $|u_k$, v$|$)

- 以上完全符合PFS的框架，唯一要做的工作无非是

  按照prioUpdater()模式，编写一个优先级（数）更新器...

实现

❖ `g->pfs( 0, PrimPU<char, int>() ); //从顶点0出发，启动Prim算法`

❖ template <typename Tv, typename Te> //顶点类型、边类型

```
template <typename Tv, typename Te> //顶点类型、边类型
struct PrimPU { //Prim算法的顶点优先级更新器
   virtual void operator()( Graph<Tv, Te>* g, int uk, int v ) {
      if ( UNDISCOVERED != g->status(v) ) return;
   // 对uk的每个尚未被发现的邻居v，按Prim策略做松弛
      if ( g->priority(v) > g->weight(uk, v) ) {
         g->priority(v) = g->weight(uk, v);
         g->parent(v)   = uk;
      }
   }
};
```